Journal on Satisfiability, Boolean Modeling, and Computation 11 (2019) 155-210

CAQE and QuAbS: Abstraction Based QBF Solvers

Leander Tentrup[*] tentrup@react.uni-saarland.de
Reactive Systems Group, Saarland University

Abstract

We present a detailed description, analysis, and evaluation of the clausal abstraction approach for solving quantified Boolean formulas (QBF). The clausal abstraction algorithm started as a solving algorithm for QBFs in prenex conjunctive normal form (PCNF) incorporating an efficient Skolem and Herbrand function extraction. Extracting witnesses from solving is especially important as it enables the certification of the solver's verdict and it is the foundation for applications built on QBF, like verification and synthesis. Later, the algorithmic ideas were extended to non-prenex and negation normal form formulas, leading the way for improved performance in solving and function extraction. The implementations of the algorithms in the solvers CAQE and QuAbS won the QBF competition (QBFEVAL'18) in their respective categories, prenex CNF and prenex non-CNF.

Keywords: *QBF solver, certification, conjunctive normal form, negation normal form*

Submitted November 2018; revised April 2019; published September 2019

1. Introduction

Efficient solving techniques for quantified logics are a prerequisite towards scalable synthesis algorithms. In contrast to verification, where the implementation is given, synthesis constructs correct-by-design implementations from a formal specification. While verification amounts to solving a one-player game ("does there exist a counterexample"), synthesis algorithms can be usually formulated as a variation of a two-player game: one player trying to satisfy the specification and an opponent player trying to falsify it. The degree of freedom and the hierarchy of information in such games—the players may choose their action based on their own memory structure and the actions of the opponents—leads to propositional problems of enormous size. Quantified Boolean formulas (QBF) have been repeatedly considered as a solving target for synthesis algorithms [9,15,16,23,26–29,58,77] and there exists evidence that quantified logics can be used to improve scalability of such synthesis methods [23]. A quantified Boolean formula is an extension of propositional logic with quantification over Boolean variables. Thus, the satisfiability problem becomes a game between two players as well: the existential player, trying to satisfy the formula, and the universal player, trying to falsify it.

Clausal Abstraction is a solving method for quantified Boolean formulas that was independently developed by Janota & Marques-Silva [46][1] and Rabe & Tentrup [66]. While

∗ This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center "Foundations of Perspicuous Software Systems' (TRR 248, 389792660) and by the European Research Council (ERC) Grant OSARES (No. 683300).

1. which they called *clause selection*

initially only applicable to QBFs in prenex conjunctive normal form, there have been extensions to QBFs in negation normal form [36], parallelization [71], satisfiability modulo theories [13], quantified stochastic Boolean satisfiability [51], and dependency quantified Boolean formulas [73]. The underlying idea of clausal abstraction is to assign variables, where the assignment order is determined by the quantifier prefix, until either all clauses are satisfied or there is a set of clauses that cannot be satisfied at the same time. The effect of assignments, i.e., whether they satisfy a clause, is abstracted into one bit of information per clause and this information is communicated through the quantifier prefix. The fundamental data structure of the algorithm is an abstraction, a propositional formula for each maximal block of quantifiers, that, given the valuation of outer variables, generates candidate assignments for the variables bound at this quantifier block. In case this candidate is refuted by inner quantifiers, the returned counterexample is excluded in the abstraction. Thus, the clausal abstraction algorithm uses ideas of search-based solving [30] and counterexample guided abstraction refinement (CEGAR) algorithms [20]. A proof theoretic analysis of the clausal abstraction approach [72] has shown that the refutation proofs correspond to the (level-ordered) Q-resolution calculus [49]. The implementation of the clausal abstraction algorithm in the solver CAQE won the prenex CNF track in the annual QBF competition QBFEVAL [59,63] 2017, 2018, and 2019. Further, it was awarded a medal in the FLoC Olympic Games 2018[2].

Beyond conjunctive normal form (CNF), there have been many attempts to improve solving performance by going to more general formula representations, such as circuits [22, 32, 34, 50]. Those approaches close the gap in expressive power between universal and existential players in CNF [47] and often outperform CNF-based solvers on practical benchmarks. We present an extension of the clausal abstraction algorithm to QBFs in negation normal form (NNF). The implementation in the solver QuAbS is used in the reactive synthesis tool BoSy [24], the Petri game solver Adam [26] and the HyperLTL satisfiability solver MGHyper [27]. Also, QuAbS won the prenex non-CNF track of QBFEVAL 2018 as well as 2019 and was awarded a medal in the FLoC Olympic Games 2018.

This article gives a complete overview over the clausal abstraction approach for QBF and is partially based on prior published work [36, 66, 72]. The remainder of this article is structured as follows. After presenting the necessary preliminaries in Section 2, we give the algorithmic details for the clausal abstraction algorithm, first for the one-alternation fragment of QBF in Section 3 followed by the generalization to quantified Boolean formulas with arbitrary many quantifiers in Section 4. In Section 5 we show how function extraction is realized and in Section 6 we integrate partial expansion reasoning in the clausal abstraction approach. The negation normal form algorithm is presented in Section 7, followed by an experimental evaluation of the CNF and NNF algorithms, implemented in the solvers CAQE and QuAbS, respectively, in Section 8. We conclude with Section 9.

Related Work. QBF solving techniques can be roughly characterized into search-based and expansion-based methods. Solvers based on search assign variables in the order given by the quantifier prefix and progress by learning clauses and cubes for conflicts and solutions, respectively. Expansion-based solving methods eliminate quantifiers by rewriting the formula into propositional form. On the algorithmic side, many recent solving meth-

2. http://www.floc2018.org/floc-olympic-games/

ods [14, 41, 42, 44, 46, 66] employ a variant of the CEGAR [20] style of reasoning to avoid exponential blowup.

Search-based Solving. Search-based solvers typically extend algorithms for the propositional satisfiability (SAT) problem to the richer logic. An early example for such an extension are the algorithms implemented in the solvers Quaffle [76] and QUBE++ [31]. The proof system underlying search-based solvers is Q-resolution [49], which extends propositional resolution with universal reduction. A more recent solver is DepQBF [54, 56], which features a variety of other extensions such as Skolem and Herbrand function extraction [61], incremental solving [55], and inprocessing [52]. Qute [62] is a search-based solver that learns dependencies between variables during the execution. The clausal abstraction approach [66], respectively, clause selection [46], can be characterized as search-based as they assign variables contained in quantifier blocks simultaneously using a SAT oracle. While the difference between the basic algorithms of clausal abstraction and clause selection is minor [46, 66], there are a number of algorithmic improvements described for clausal abstraction [66, 71] that make the implementation CAQE outperform the clause selection solver Qesto as shown in the evaluation in Section 8.1.

There are further extensions of search-based methods to quantified Boolean formulas beyond conjunctive normal form [22, 34, 50, 62]. These methods typically exploit the duality of propositional formulas in negation normal form. Further approaches include using antichains as the underlying data structure [17] and using the duality of negation normal form to enhance CNF solving [35]. The clausal abstraction approach has been generalized to QBFs in negation normal form [36] and to non-prenex formulas [71]. cQESTO [41] is a recently introduced circuit solver based on a similar algorithm as presented in this article. 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.

Recently, incremental determinization [65, 67] has been proposed as a search-based algorithm whose propagation mechanism is based on Boolean functions instead of variable assignments.

Expansion-based Solving. For expansion-based methods, one can further distinguish into complete and partial expansion. Complete expansion eliminates all universal quantifiers and rewrites the QBF to an equisatisfiable propositional formula. Design choices include the order of elimination, rewriting, and the representation of propositional formulas. Examples for complete expansion solvers are Qubos [1], Quantor [11], Nenofex [53], and AIGSolve [68]. dynQBF [19] is a recent solver that traverses a tree decomposition of a QBF instance and uses dynamic programming in conjunction with BDDs to solve subproblems.

Partial expansion tries to expand only a subset of the possible universal assignments in order to show unsatisfiability (and dually, satisfiability). RAReQS [44] is a solver based on partial expansion that has later been extended to include refinements with strategies [42]. The underlying proof system, $\forall$Exp+Res [45], first builds a partial expansion of the QBF and then uses propositional resolution on the expanded matrix. Recently, an algorithm based on partial expansion called Ijtihad [14] was proposed that uses only two competing SAT solvers, whereas RAReQS uses one per quantifier block in the prefix.

Hybrid Approaches. Hybrid approaches combine both, search-based and expansion-based reasoning, with different levels of integration. The search-based solver GhostQ [50] incorporates partial expansion reasoning [44]. Heretic [14] is a lightweight integration of Ijtihad and DepQBF. The clausal abstraction solver CAQE has been extended to include partial expansion reasoning [72]. What makes the hybrid approaches theoretically appealing and in practice performant is the fact that the proof systems underlying search, Q-resolution, and partial expansion, $\forall$Exp+Res, are incomparable with respect to polynomial simulation [10, 45], that is, neither does Q-resolution subsume $\forall$Exp+Res nor vice versa. Hence, a solver that combines both types of reasoning has a potential advantage over both, expansion and search-based solvers [72].

Preprocessing. Whereas this article is only concerned with complete solving techniques for quantified Boolean formulas, there is a rich body of literature regarding QBF preprocessing techniques. Further, our experiments in Section 8.1 show that preprocessing is an integral part of the performance characteristics of modern clausal QBF solvers and this applies to clausal abstraction as well.

Blocked clause elimination is a common preprocessing technique, implemented (among other preprocessing techniques) in the tool Bloqqer [12]. HQSPre [75] is a preprocessor for both, QBF and DQBF. Both also use (incomplete) universal expansion as well as variable elimination using resolution as preprocessing techniques. Recently, a new preprocessor QRATPre+ [57] was introduced, that is based on the QRAT calculus [37].

Certification and Function Extraction. The need for providing solving witnesses beyond binary answers is a research question that started with the very first QBF solving algorithms. The solver sKizzo [6] is one of the earliest QBF solver that included certification [8]. An early certification format was proposed by Jussila et al. [48] and implemented for the solvers Quaffle and Squolem. Balabanov and Jiang [2] showed how to extract Skolem and Herbrand functions from term-resolution and Q-resolution proofs, respectively. The QBFCert framework [61] is an implementation of this approach for the search-based solver DepQBF [56]. There have been further improvements to the extraction algorithm [5] and extensions to handle long-distance resolution [3, 21]. As long as there were solvers with certification capabilities, there are attempts to provide a unified framework [7, 48, 70] with the most recent one, QRAT [37], being the most promising as it was already successfully applied to preprocessing [38]. To overcome the problem of missing preprocessing in the context of certification, there has been work that combines certificates produced by solver and preprocessor [25, 43]. For non-CNF solvers, there have also been methods for extracting Skolem and Herbrand functions [17, 33].

2. Quantified Boolean Formulas

2.1 Syntax

A *quantified Boolean formula* (QBF) [18] is a propositional formula over a finite set of variables $\mathcal{V}$ with Boolean domain $\mathbb{B} = \{\mathbf{F}, \mathbf{T}\}$ and quantification over variables. The syntax is given by the grammar

$$\varphi \coloneqq v \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists v.\, \varphi,$$

where $v \in \mathcal{V}$. Let $\mathcal{B}(V)$ be the set of quantified Boolean formulas over variables V. We use the usual Boolean connectives *conjunction* $\varphi \wedge \psi \coloneqq \neg(\neg\varphi \vee \neg\psi)$, *implication* $\varphi \to \psi \coloneqq \neg\varphi \vee \psi$, *equivalence* $\varphi \leftrightarrow \psi \coloneqq (\varphi \to \psi) \wedge (\psi \to \varphi)$, and *exclusion* $\varphi \oplus \psi \coloneqq \neg(\varphi \leftrightarrow \psi)$. Universal quantification $\forall v.\, \varphi$ is defined as $\neg\exists v.\, \neg\varphi$.

We denote the universally and existentially quantified variables as *universals* and *existentials*, respectively. To improve readability, we lift the consecutive quantification over variables of the same type to the quantification over sets of variables and denote $\mathcal{Q}v_1 \ldots \mathcal{Q}v_n.\, \varphi$ by $\mathcal{Q}V.\, \varphi$ for $V = \{v_1, \ldots, v_n\}$ and $\mathcal{Q} \in \{\forall, \exists\}$. We assume w.l.o.g. that every variable $v \in \mathcal{V}$ is quantified at most once. A quantifier block $\mathcal{Q}v.\, \varphi$ for $\mathcal{Q} \in \{\exists, \forall\}$ *binds* the variable v in the *scope* φ. Variables that are not bound by a quantifier are called *free*. We refer to the set of free variables of formula φ as *free*(φ). A *closed* QBF is a formula without free variables. Closed QBFs are either true or false. Every QBF can be transformed into a closed QBF while maintaining satisfiability by prepending the formula with existential quantifiers that bind the free variables. A formula is in prenex form, if the formula consists of a quantifier prefix followed by a propositional, i.e., quantifier-free, formula. Every QBF can be transformed into prenex form while maintaining satisfiability. For a $k > 0$, a formula φ is in the kQBF fragment if it is closed, in prenex form, and has exactly $k-1$ alternations between $\exists$ and $\forall$ quantifiers.

A *literal* l is either a variable $v \in V$, or its negation $\neg v$. The complement of a literal l, written $\bar{l}$, is defined as $\bar{l} = \neg v$ if $l = v$, and $\bar{l} = v$ if $l = \neg v$. Given a literal $l = v$ or $l = \neg v$, we define *var*$(l) = v$. 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*.

A QBF is in *prenex conjunctive normal form* (PCNF) if its propositional formula is a conjunction over clauses, i.e., in conjunctive normal form (CNF). We call the propositional part of a QBF in PCNF the *matrix* and we use C_i to refer to clause i of the matrix where unambiguous. For convenience, we treat clauses and matrices as a sets of literals and clauses, respectively, and use the usual set operations for their manipulation. When given matrices, we typically omit the $\wedge$ operator between clauses. Every QBF in prenex form can be transformed into an equisatisfiable formula in PCNF using the Tseitin transformation [74] with a linear increase in the size of the formula and number of existential variables.

Example 1. The following quantified Boolean formula

$$\exists v, w.\, \forall x.\, \exists y, z.\, (w \vee x \vee y)(v \vee \overline{w})(x \vee \overline{y})(\overline{v} \vee z)(\overline{z} \vee \overline{x})$$

is in the 3QBF fragment and its propositional part is in conjunctive normal form.

A QBF is in *negation normal form* (NNF) if negation is only applied to variables, that is, a formula in NNF contains only conjunctions, disjunctions, and literals. Every QBF can be transformed into NNF by at most doubling the size of the formula and without introducing new variables as it is the case for the Tseitin transformation.

Example 2. The following quantified Boolean formula

$$\exists x.\, \forall v, w.\, \exists y.\, (x \vee v \vee (y \wedge w)) \wedge (\overline{x} \vee (v \wedge \overline{w}) \vee y) \wedge (\overline{v} \vee w \vee \overline{y})$$

has two quantifier alternations and its propositional formula is in negation normal form.

2.2 Boolean Assignments and Functions

Given a subset of variables $V \subseteq \mathcal{V}$, a *Boolean assignment* of V is a function $\alpha\colon V \to \mathbb{B}$ that maps each variable $v \in V$ to either true ($\mathbf{T}$) or false ($\mathbf{F}$). We write α_V when the domain of α, written $\mathrm{dom}(\alpha)$, is not clear from the context. A *partial assignment* $\beta\colon V \to \mathbb{B}_\perp$, where $\mathbb{B}_\perp \coloneqq \mathbb{B} \cup \{\perp\}$, may additionally set variables $v \in V$ to an undefined value $\perp$. We use the notation α^+ and α^- to denote the partial assignment that retains positive and negative variable assignments, respectively. It is defined as

$$\alpha^+(v) = \begin{cases} \alpha(v) & \text{if } \alpha(v) = \mathbf{T} \\ \perp & \text{otherwise} \end{cases} \qquad \text{and} \qquad \alpha^-(v) = \begin{cases} \alpha(v) & \text{if } \alpha(v) = \mathbf{F} \\ \perp & \text{otherwise} \end{cases}$$

for every $v \in \mathrm{dom}(\alpha)$. We use the replacement operator $\beta_V[\perp \mapsto b]$ for $b \in \mathbb{B}$ to denote the assignment where undefined is replaced by a default value b. It is defined as

$$\beta_V[\perp \mapsto b](v) \coloneqq \begin{cases} \beta_V(v) & \text{if } \beta_V(v) \neq \perp \\ b & \text{otherwise} \end{cases}$$

for every $v \in V$. To restrict the domain of an assignment α to a set of variables V, we write $\alpha|_V$. For two assignments α and α' with domains $V = \mathrm{dom}(\alpha)$ and $V' = \mathrm{dom}(\alpha')$, we define the combination $\alpha \sqcup \alpha'\colon V \cup V' \to \mathbb{B}$ as

$$(\alpha \sqcup \alpha')(v) = \begin{cases} \alpha'(v) & \text{if } v \in V' \\ \alpha(v) & \text{otherwise} \end{cases} .$$

Note that α' overrides α for every element $v \in V \cap V'$ in the intersection of their domains. If the domains of α and α' are disjoint, that is, $\mathrm{dom}(\alpha) \cap \mathrm{dom}(\alpha') = \emptyset$, we denote the combination by $\alpha \mathbin{\dot\sqcup} \alpha'$. For two partial assignments β_V and β'_V, we define the intersection operation $\beta_V \sqcap \beta'_V\colon V \to \mathbb{B}_\perp$ as

$$(\beta_V \sqcap \beta'_V)(v) = \begin{cases} \beta_V(v) & \text{if } \beta_V(v) = \beta'_V(v) \\ \perp & \text{otherwise} \end{cases} .$$

We define the *complement* $\overline{\alpha}$ to be $\overline{\alpha}(v) = \neg\alpha(v)$ for all $v \in \mathrm{dom}(\alpha)$. The complement of a partial assignment is defined analogously with $\neg\perp = \perp$. We use the notation $\varphi[\alpha]$ to replace variables $v \in \mathrm{dom}(\alpha)$ with their assignments $\alpha(v)$. We denote by $\alpha_V^b \coloneqq \{v \in V \mid \alpha_V(v) = b\}$ the subset of variables that are assigned to $b \in \mathbb{B}$, i.e., the preimage of α_V with respect to b. The *set of assignments* and the *set of partial assignments* of V is denoted by $\mathcal{A}(V)$ and $\mathcal{A}_\perp(V)$, respectively.

A *Boolean function* $f\colon \mathcal{A}(V) \to \mathbb{B}$ maps *assignments* of V to true or false. An assignment α_V over variables V can be represented by the conjunctive formula $\bigwedge_{v \in \alpha_V^{\mathbf{T}}} v \wedge \bigwedge_{v \in \alpha_V^{\mathbf{F}}} \neg v$, that is, the only assignment over variables V that satisfy this formula is the assignment α_V. Similarly, Boolean functions can be represented by propositional formulas over the variables in their domain. Let $\varphi[f_v]$ be the formula where occurrences of v are replaced by the propositional representation of f_v. It is defined as

$$\begin{aligned}
x[f_v] &= \begin{cases} f_v & \text{if } v = x \\ x & \text{otherwise} \end{cases} \\
(\neg\varphi)[f_v] &= \neg(\varphi[f_v]) \\
(\varphi \vee \psi)[f_v] &= (\varphi[f_v]) \vee (\psi[f_v]) \\
(\exists x.\, \varphi)[f_v] &= \begin{cases} \varphi[f_v] & \text{if } v = x \\ \exists x.\, (\varphi[f_v]) & \text{otherwise} \end{cases}
\end{aligned}$$

For example, let $\varphi = \forall x.\, \exists y.\, (x \vee \neg y) \wedge (\neg x \vee y)$ and let $f_y(x) = x$, then $\varphi[f_y] = \forall x.\, (x \vee \neg x) \wedge (\neg x \vee x)$. We use a function $g\colon \mathcal{A}(X) \to \mathcal{A}(Y)$ that maps assignments of X to assignments of Y to represent multiple Boolean functions and define the replacement operator $\varphi[g]$ accordingly. For example, given another Boolean function $f_{y'}\colon \mathcal{A}(\{x\}) \to \mathbb{B}$, the combination with f_y is $g_{y,y'}\colon \mathcal{A}(\{x\}) \to \mathcal{A}(\{y, y'\})$ such that $g_{y,y'}(x) = \{y \mapsto f_y(x), y' \mapsto f_{y'}(x)\}$.

2.3 Semantics

We fix a set of variables $V \subseteq \mathcal{V}$. The satisfaction relation $\vDash\ \subset \mathcal{A}(V) \times \mathcal{B}(V)$ is defined as

$$\begin{array}{ll}
\alpha \vDash v & \text{if } \alpha(v) = \mathbf{T}, \\
\alpha \vDash \neg\varphi & \text{if } \alpha \nvDash \varphi, \\
\alpha \vDash \varphi \vee \psi & \text{if } \alpha \vDash \varphi \text{ or } \alpha \vDash \psi, \text{and} \\
\alpha \vDash \exists v.\, \varphi & \text{if there exists some } \alpha'\colon \mathcal{A}(\{v\}) \to \mathbb{B} \text{ such that } \alpha \,\dot\sqcup\, \alpha' \vDash \varphi.
\end{array}$$

QBF satisfiability is the problem to determine, for a given QBF Φ, the existence of an assignment α for the free variables $\mathit{free}(\Phi)$ such that the relation $\vDash$ holds. In this case, we call α a satisfying assignment and say that α satisfies Φ. If $\alpha \nvDash \Phi$, we say that α falsifies Φ. For a closed form QBF Φ, the QBF satisfiability problem is equivalent to the validity problem, which asks if all assignments satisfy Φ, as the problem reduces to checking whether $\{\} \vDash \Phi$ where $\{\}$ denotes the empty assignment. For formulas in prenex form with propositional formula φ, the QBF satisfiability problem can be interpreted as a two-player game: Based on the order of quantifiers given by the quantifier prefix, the existential player $\exists$ chooses assignments of existential variables with the aim to satisfy φ, while the universal player $\forall$ chooses assignment of universal variables in order to falsify φ. The satisfiability game is *determined*, that is, for every QBF, either the existential player or the universal player has a winning strategy.

For satisfiable QBFs the winning strategy for the existential player is called a *Skolem function* $f\colon \mathcal{A}(V_\forall) \to \mathcal{A}(V_\exists)$ which maps assignments of universal variables $V_\forall$ to assignments of existential variables $V_\exists$, such that $\varphi[f]$ is valid. For unsatisfiable QBFs, the winning strategies are defined dually, i.e., $f\colon \mathcal{A}(V_\exists) \to \mathcal{A}(V_\forall)$ such that $\varphi[f]$ is unsatisfiable, and are called *Herbrand functions*. Intuitively, Skolem and Hebrand functions are well-formed if every assigned variable depends solely on its dependencies as given by the quantifier prefix. We formalize this intuition in the following using the concept of *dependencies* and *consistency*.

An existentially quantified variable v *depends* on all universally quantified variables that are bound prior to v. A universally quantified variable v *depends* on all existentially quantified variables bound prior to v as well as the free variables. A free variable v has no dependencies, i.e., can only be instantiated by constants. The set of dependencies of a variable $v \in V$ is denoted by $dep(v)$. For a set of variables V, we define $dep(V)$ as the union over the dependencies $\bigcup_{v \in V} dep(v)$.

A function f_X is *well-formed* if the assignments are *consistent* with respect to the dependencies of X, i.e., for every $x \in X$ and every pair of assignments α_V and α'_V with $V = dep(X)$ and $\alpha_V|_{dep(x)} = \alpha'_V|_{dep(x)}$ it holds that $f_X(\alpha_V)(x) = f_X(\alpha'_V)(x)$. In other words, f_X has to produce the same output for $x \in X$ if the dependencies of x are the same.

3. Solving QBF with One Quantifier Alternation

We start the description of the clausal abstraction algorithm by considering only the one-alternation fragment of QBF, called 2QBF. In this fragment, the existential variables have *complete information*, i.e., they depend on the complete set of universal variables. The reasons for choosing 2QBF as a starting point are manifold; it is in some sense the simplest extension of propositional logic that includes quantification and allows us to introduce the core ideas, notation, and terminology behind the clausal abstraction algorithm. After discussing the restricted fragment, we generalize the algorithm to arbitrary quantifier alternations in Section 4. For this section, we fix some 2QBF $\forall X. \exists Y. \varphi$ with universal variables X, existential variables Y, and matrix φ.

3.1 Algorithm

Preliminaries. We use a generic solving function $\textsc{sat}(\theta, \alpha)$ for propositional formula θ and assignment α, that returns whether $\theta \wedge \alpha$ is satisfiable. In the positive case, it returns $\mathsf{Sat}(\alpha')$, where α' is a satisfying assignment of θ with $\alpha \sqsubseteq \alpha'$. In the negative case, it returns $\mathsf{Unsat}(\beta)$, where $\beta \sqsubseteq \alpha$ is a partial assignment such that $\theta \wedge \beta$ is unsatisfiable.

In the following algorithms, we make use of pattern matching on well-structured objects, such as the result of the call to SAT and the quantifier prefix of quantified Boolean formulas. For example, to determine the leading quantifier of some QBF Φ, we write

match Φ **as**
  $\exists X. \Psi \Rightarrow [\dots]$ ▷ leading existential quantifier
  $\forall X. \Psi \Rightarrow [\dots]$ ▷ leading universal quantifier

Additionally, we allow wildcards, denoted by "_", in match arms.

Overview. The clausal abstraction algorithm is based on the idea of using two competing SAT solvers, one for the universal quantifier that tries to falsify clauses and one for the existential quantifier that has to satisfy the remaining clauses in the matrix. The algorithm $\textsc{solve}_{\forall\exists}$ is shown in Algorithm 1. After initializing the abstractions, which is detailed below, the algorithm repeatedly solves θ_X using a SAT solver. θ_X contains variables X and satisfaction variables S, one variable $s_i \in S$ for every clause $C_i \in \varphi$ that represents whether this clause is satisfied by an assignment α_X of variables X. Every assignment α with $\alpha \vDash \theta_X$ is a combination of an assignment $\alpha|_X$ of variables X and an assignment $\alpha|_S$

Algorithm 1 Clausal Abstraction Algorithm for 2QBF

procedure $\textsc{solve}_{\forall\exists}(\forall X.\, \exists Y.\, \varphi)$
  initialize abstractions θ_X and θ_Y with shared variables $S = \{s_i \mid C_i \in \varphi\}$
  loop
    match $\textsc{sat}(\theta_X, \{\})$ **as**
      $\mathsf{Unsat}(\_) \Rightarrow$ **return** Sat
      $\mathsf{Sat}(\alpha) \Rightarrow$
        match $\textsc{sat}(\theta_Y, \alpha|_S)$ **as**
          $\mathsf{Unsat}(\_) \Rightarrow$ **return** $\mathsf{Unsat}(\alpha|_X)$ ▷ $\alpha|_X \nvDash \exists Y.\, \varphi$
          $\mathsf{Sat}(\_) \Rightarrow \theta_X \leftarrow \theta_X \wedge \bigvee_{s \in (\alpha|_S)^{\mathbf{T}}} \overline{s}$ ▷ refine θ_X
  end loop
end procedure

of variables S. In the following SAT call to θ_Y, the assignment $\alpha|_S$ representing satisfied clauses is assumed. In case $\theta_Y[\alpha|_S]$ is satisfiable, we found a matching Y assignment to the given X assignment, thus, the abstraction θ_X is refined and the algorithm proceeds with the next iteration. The algorithm terminates, returning satisfiable and unsatisfiable, if the SAT call to θ_X and θ_Y is unsatisfiable, respectively. In the former case, we have depleted all universal assignments and in the latter case there is an assignment α_X such that there is no matching Y assignment.

Abstractions $\boldsymbol{\theta_X}$ and $\boldsymbol{\theta_Y}$. The abstraction is the core data structure of the algorithm, representing, for each player, an over-approximation of the winning assignments and the resulting effect those assignments have on the satisfaction of clauses. The abstraction θ_Y represents the winning assignments α_Y of the existential player under the condition that a certain set of clauses is already satisfied by the prior universal assignment α_X. Thus, θ_Y is satisfiable if, and only if, every clause in the matrix is satisfied, either by an assignment to Y or by an assignment of the outer universal variables. For universal quantifier $\forall X$, the abstraction θ_X represents which clauses are satisfied with respect to an assignment to X. During the execution of the algorithm, we learn that the universal player cannot falsify φ when a certain set of clauses is satisfied by α_X, thus, we refine θ_X to make sure that one of the previously satisfied clauses is falsified, which eliminates losing assignments α_X.

The interaction between θ_X and θ_Y is established by a common set of *clause satisfaction* variables S, one variable $s_i \in S$ for every clause $C_i \in \varphi$. Given an assignment α_X and some clause $C_i \in \varphi$, we guarantee that s_i is assigned to true if $\alpha_X \vDash C_i|_X$. Thus, if s_i is assigned to false, the existential quantifier has to satisfy clause C_i. We define the abstraction that implements those requirements for a clause $C_i \in \varphi$ as

$$clabs_{\forall X}(C_i) \coloneqq s_i \vee \neg C_i|_X = \bigwedge_{l \in C_i|_X} \bar{l} \vee s_i \text{ and} \tag{1}$$

$$clabs_{\exists Y}(C_i) \coloneqq s_i \vee C_i|_Y \tag{2}$$

for universal and existential quantifier, respectively. The clausal abstraction for the universal quantifier block $\forall X$ and the existential quantifier block $\exists Y$ is defined as

$$\theta_X \coloneqq \bigwedge_{C_i \in \varphi} \mathit{clabs}_{\forall X}(C_i) \qquad \text{and} \qquad \theta_Y \coloneqq \bigwedge_{C_i \in \varphi} \mathit{clabs}_{\exists Y}(C_i) \ . \tag{3}$$

Lastly, a refinement for θ_X ensures that from a set of clauses that was previously satisfied (s_i set to true) one is falsified, thus we add the clause

$$\bigvee_{s \in \alpha_S^{\mathbf{T}}} \overline{s} \tag{4}$$

to the abstraction θ_X. We conclude the description of the algorithm by a detailed example. In the following section, we show that the algorithm correctly determines the result of the satisfiability problem for 2QBF.

Example 3. Consider the following 2QBF

$$\forall x.\, \exists y, z.\, (x \vee z)(\overline{x} \vee \overline{y})(\overline{x} \vee y \vee z)(\overline{z} \vee \overline{x}).$$

By the definitions above, the resulting abstractions are

$$\begin{aligned} \theta_{\{x\}} &= (s_1 \vee \overline{x})(s_2 \vee x)(s_3 \vee x)(s_4 \vee x) \text{ and} \\ \theta_{\{y,z\}} &= (s_1 \vee z)(s_2 \vee \overline{y})(s_3 \vee y \vee z)(s_4 \vee \overline{z}) \ . \end{aligned}$$

We show a possible execution of $\textsc{solve}_{\forall\exists}$ on the example formula:

- $\textsc{sat}(\theta_{\{x\}}, \{\}) = \mathsf{Sat}(\{x \mapsto \mathbf{F}, s_1 \mapsto \mathbf{F}, s_2 \mapsto \mathbf{T}, s_3 \mapsto \mathbf{T}, s_4 \mapsto \mathbf{T}\})$
- $\textsc{sat}(\theta_{\{y,z\}}, \{s_1 \mapsto \mathbf{F}, s_2 \mapsto \mathbf{T}, s_3 \mapsto \mathbf{T}, s_4 \mapsto \mathbf{T}\}) = \mathsf{Sat}(\{z \mapsto \mathbf{T}, y \mapsto \mathbf{F}\})$
- $\theta'_{\{x\}} = \theta_{\{x\}} \wedge (\overline{s_2} \vee \overline{s_3} \vee \overline{s_4})$
- $\textsc{sat}(\theta'_{\{x\}}, \{\}) = \mathsf{Sat}(\{x \mapsto \mathbf{T}, s_1 \mapsto \mathbf{T}, s_2 \mapsto \mathbf{F}, s_3 \mapsto \mathbf{F}, s_4 \mapsto \mathbf{F}\})$
- $\textsc{sat}(\theta_{\{y,z\}}, \{s_1 \mapsto \mathbf{T}, s_2 \mapsto \mathbf{F}, s_3 \mapsto \mathbf{F}, s_4 \mapsto \mathbf{F}\}) = \mathsf{Unsat}$
- $\textsc{solve}_{\forall\exists}$ returns $\mathsf{Unsat}(\{x \mapsto \mathbf{T}\})$

3.2 Correctness

The correctness argument relates variable assignments to assignments of the satisfaction variables S. We start by stating two properties over the abstractions θ_X and θ_Y that immediately follow from their definitions.

Lemma 1. *Let α be a satisfying assignment of θ_X and let α_S be some arbitrary assignment over variables S. It holds that*

1. $\alpha(s_i) = \mathbf{F} \Rightarrow \alpha|_X \nvDash C_i|_X$ *for every clause* $C_i \in \varphi$ *and*
2. $\theta_Y[\alpha_S] = \bigwedge_{s_i \in \alpha_S^{\mathbf{F}}} C_i|_Y$.

For termination, we need to argue that the main loop in Algorithm 1 cannot be executed infinitely often. We give an implicit ranking function, based on the following observations. First, the number of different refinements, i.e., clauses over S, is bounded by the number of variables in S. Second, during the execution of the algorithm, every refinement clause (line 9) is different, that is, it is impossible that two refinements are the same.

Lemma 2. *There are only finitely many different refinement clauses and the refinements during the execution of Algorithm 1 are pairwise different.*

Proof. The number of different refinement clauses is bounded by the number of subsets of S by the definition in Equation 4, i.e., are at most $2^{|S|}$ different refinement clauses. Assume for contradiction that there is an execution of the algorithm that produces the same refinement clause R, thus, according to line 9 of Algorithm 1 there are two assignments α and α' such that $(\alpha|_S)^1 = (\alpha'|_S)^1$. It holds that $R = \bigvee_{s \in (\alpha|_S)^{\mathbf{T}}} \overline{s}$ and thus $\alpha' \nvDash R$. As θ_X contains the clause R after the refinement with α and α' satisfies θ_X, we derive a contradiction. □

Given those lemmas, we can prove the correct termination for true formulas.

Theorem 1. *If $\forall X.\, \exists Y.\, \varphi$ is true, Algorithm 1 returns* Sat.

Proof. Let $\forall X.\, \exists Y.\, \varphi$ be true. By the QBF semantics, there is a Skolem function $f_Y : \mathcal{A}(X) \to \mathcal{A}(Y)$ such that $\varphi[f_Y]$ is valid. Let α_X and α_S be arbitrary assignments satisfying θ_X (line 4). By Lemma 1, it holds that

$$\theta_Y[\alpha_S] = \bigwedge_{s_i \in \alpha_S^{\mathbf{F}}} C_i|_Y \quad \subseteq \quad \bigwedge_{\substack{C_i \in \varphi \\ \alpha_X \nvDash C_i|_X}} C_i|_Y = \varphi[\alpha_X] \ .$$

Hence, $\alpha_Y \coloneqq f_Y(\alpha_X)$ is a satisfying assignment for $\theta_Y[\alpha_S]$ as it satisfies $\varphi[\alpha_X]$. The formula $\theta_Y[\alpha_S]$ in line 7 is, thus, always satisfiable and the return in line 8 is unreachable. Termination is guaranteed by Lemma 2. □

For the reverse direction, we need additional properties regarding the refinement operation that we state in the following. Let α_X be an assignment and let θ_X and θ'_X be the abstraction before and after the refinement with some assignment α_S, respectively. We say that α_X is *excluded* from θ_X if $\theta'_X[\alpha_X]$ is unsatisfiable whereas $\theta_X[\alpha_X]$ is satisfiable.

Lemma 3. *If an assignment α_X is excluded from θ_X by a refinement with α_S, it holds that $\alpha_S(s_i) = \mathbf{T}$ implies that $\alpha_X \vDash C_i|_X$ for every $C_i \in \varphi$.*

Proof. Let α_X and α_S be assignments such that α_X is excluded from θ_X by a refinement with α_S, that is, $\theta_X[\alpha_X]$ is satisfiable and the refinement clause $\psi = \bigvee_{s \in \alpha_S^{\mathbf{T}}} \overline{s}$ (line 9 of Algorithm 1) excludes α_X, i.e., $\theta'_X = \theta_X \wedge \psi$ and $\theta'_X[\alpha_X]$ is unsatisfiable. θ'_X entails $\psi' \coloneqq \bigvee_{s_i \in \alpha_S^{\mathbf{T}}} \neg C_i|_X$ (see the definition of the universal abstraction in Equation 1) and it holds that $\alpha_X \nvDash \psi'$ (assuming otherwise would contradict that $\theta'_X[\alpha_X]$ is unsatisfiable). Thus, it holds that $\alpha_X \vDash \bigwedge_{s_i \in \alpha_S^1} C_i|_X$. □

Theorem 2. *If $\forall X.\, \exists Y.\, \varphi$ is false, Algorithm 1 returns* Unsat(α_X) *where $\varphi[\alpha_X]$ is unsatisfiable.*

Proof. Let $\forall X. \exists Y. \varphi$ be false. By the QBF semantics, there exists some assignment α_X such that $\varphi[\alpha_X]$ is unsatisfiable. Let α_S be the assignment such that $\alpha_S(s_i) = \mathbf{T}$ if, and only if, $\alpha_X \vDash C_i|_X$ for every $C_i \in \varphi$. The combined assignment $\alpha_S \dot{\sqcup} \alpha_X$ is a satisfying assignment for θ_X in line 4. It holds that $\theta_Y[\alpha_S] = \bigwedge_{s_i \in \alpha_S^{\mathbf{F}}} C_i|_Y = \varphi[\alpha_X]$ by the definition of the existential abstraction. As $\varphi[\alpha_X]$ is unsatisfiable, $\theta_Y[\alpha_S]$ is unsatisfiable as well, leading to the return in line 8.

To conclude the proof, it remains to show that this α_X is not excluded by some refinement in line 9. Assume for contradiction that it is excluded by some assignment α_S, i.e., by Lemma 3 for every $C_i \in \varphi$ it holds that $\alpha_S(s_i) = \mathbf{T} \Rightarrow \alpha_X \vDash C_i|_X$ which is equivalent to $\alpha_X \nvDash C_i|_X \Rightarrow \alpha_S(s_i) = \mathbf{F}$. It holds that

$$\varphi[\alpha_X] = \bigwedge_{\substack{C_i \in \varphi \\ \alpha_X \nvDash C_i|_X}} C_i|_Y \quad \subseteq \quad \bigwedge_{s_i \in \alpha_S^{\mathbf{F}}} C_i|_Y = \theta_Y[\alpha_S] \ ,$$

thus, from the unsatisfiability of $\varphi[\alpha_X]$ follows that $\theta_Y[\alpha_S]$ is unsatisfiable as well, contradicting the refinement of α_S. As there are only finitely many different refinements, the query in line 4 eventually returns the assignment α_X or some other unsatisfying assignment. □

3.3 Optimizations

In this section, we investigate improvements to the algorithm stated in Algorithm 1. Those improvements fall in two categories. The first category is concerned with simplifying the propositional abstractions with the intention to improve the satisfiability check and the second category is concerned with potentially reducing the number of iterations of the algorithm.

Abstraction Improvements. In the case that a clause $C_i \in \varphi$ contains only existential quantified variables, s_i can be assumed to be false. Thus, we can modify the definitions of clabs_Q, given in Equation 1 and Equation 2 to $\mathit{clabs}_\forall(X, C_i) = \mathbf{T}$ and $\mathit{clabs}_\exists(Y, C_i) = C_i$ if $C_i|_Y = C_i$. Note that in this case, the variable s_i does not appear in the abstraction.

Balabanov et al. [4] describe two further simplifications for the universal abstraction:

- If some clause $C_i \in \varphi$ is a universal unit clause, i.e., $C|_X = \{l\}$ for some literal l with $\mathit{var}(l) \in X$, then the shared variable s_i can be replaced by the negation $\bar{l}$ of the literal.

- If there is a pair of clauses $C_i, C_j \in \varphi$ with $i \neq j$ such that those clauses are equal with respect to universal literals, i.e., $C_i|_X = C_j|_X$, then the same shared variable s_i can be used for both clauses.

Lastly, one can use the knowledge of the objective of the universal quantifier to improve assignments α_S. As the variables S occur pure in the abstraction θ_X, the SAT solver may set all of them to false initially. For SAT solver that support setting a default polarity on decisions, this can be used to improve the initial assignment. Alternatively, the problem could be reformulated as a maximum satisfiability (MaxSAT) optimization problem.

Algorithmic Improvements. Given assignments α_S and α_X from the SAT solver in line 4, α_S may not be optimal in the following sense: there can be some clause $C_i \in \varphi$

where $\alpha_S(s_i) = \mathbf{T}$ but the assignment α_X does not satisfy $C_i|_X$, i.e., $\alpha_X \vDash \neg C_i|_X$. This is due to the implication in the definition of $\mathit{clabs}_\forall$ in Equation 1. We circumvent this by applying a step after line 4 that optimizes α_S with respect to α_X, i.e., we set $\alpha_S(s_i) = \mathbf{F}$ for every clause $C_i \in \varphi$ where $\alpha_X \vDash \neg C_i|_X$. This change is also compatible with the correctness proof in the previous section, especially Lemma 1 still holds after the α_S optimization.

Given satisfying assignments α_S and α_Y from the SAT solver in line 7, the assignment α_Y may satisfy clauses that are not required by the assignment α_S, that is, the clause is already satisfied by the universal variable assignment represented by α_S. This is the case if there is some clause $C_i \in \varphi$ with $\alpha_Y \vDash C_i|_Y$ and $\alpha_S(s_i) = \mathbf{T}$. We use this information to improve the refinement clause in Equation 4: For every such clause C_i, we set $\alpha_S(s_i) = \mathbf{F}$, thus, reducing the number of literals in the refinement clause.

Another enhancement greedily flips variable assignments in α_Y if the resulting assignment satisfies strictly more clauses. Let $y \in Y$ be some existential variable. We can flip the value of y if $\varphi[\alpha_Y \sqcup \{y \mapsto \neg\alpha_Y(y)\}] \subseteq \varphi[\alpha_Y]$. This greedy flipping may further improve the effect of the previous optimization.

Balabanov et al. [4] noticed that under certain conditions, one can remove literals from the refinement clause in Equation 4. If there are two clauses C_i and C_j with $C_i|_X \subseteq C_j|_X$ and $\alpha_S(s_i) = \alpha_S(s_j) = \mathbf{T}$, then we can set $\alpha_S(s_j) = \mathbf{F}$, which removes $\overline{s_j}$ from the refinement clause. This is due to the implications $\overline{s_i} \rightarrow \neg C_i|_X$ and $\overline{s_j} \rightarrow \neg C_j|_X$ in the universal abstraction, $\mathit{clabs}_\forall$ in Equation 1, and the implication $\neg C_j|_X = \bigwedge_{l \in C_j|_X} \neg l \Rightarrow \bigwedge_{l \in C_i|_X} \neg l = \neg C_i|_X$ due to the fact that the literals in $C_j|_X$ are a superset of the literals in $C_i|_X$.

The algorithmic optimizations are crucial for the performance of the algorithm as they circumvent non-optimal assignments to satisfaction variables resulting from using implications in the definition of the abstraction (compared to the equivalences used in clause selection [46]).

4. Solving QBF with Arbitrary Quantifier Alternations

We now generalize the 2QBF algorithm to QBFs with an arbitrary number of alternations by providing an algorithm that does recursion on the quantifier prefix. The main insight in this generalization is that the existential player now has a choice to either directly satisfy a clause or assume that an inner quantifier block will satisfy it. For this section, we fix some quantified Boolean formula Φ in closed prenex conjunctive normal form (PCNF) with matrix φ. We assume that Φ is universally reduced, that is, for every clause $C_i \in \varphi$ and every universal literal $l_\forall \in C_i$, there is an existential literal $l_\exists \in C_i$ that depends on $l_\forall$, formally $\mathit{var}(l_\forall) \in \mathit{dep}(\mathit{var}(l_\exists))$. If this property is violated for some clause C_i and literal $l_\forall \in C_i$, then $l_\forall$ can be removed from C_i, which is called universal reduction [49].

4.1 Algorithm

Overview. The overall approach of the algorithm is to construct a propositional formula θ_X for every quantifier block $\mathcal{Q}X$ that represents an over-approximation of the winning assignments α_X and the effect of those assignments on the matrix, that is, which clauses are satisfied and falsified for existential and universal quantifiers, respectively. The main algorithm SOLVE is depicted in Algorithm 2. It takes as an input a quantified Boolean

Algorithm 2 Clausal Abstraction Algorithm for QBF

```
procedure SOLVE(Φ)
    initialize abstraction θ_X for every quantifier block QX in Φ
    match Φ as
        ∃X. Ψ ⇒ return SOLVE_∃(X, Ψ, {s_i ↦ F | C_i ∈ φ})
        ∀X. Ψ ⇒ return SOLVE_∀(X, Ψ, {s_i ↦ F | C_i ∈ φ})
end procedure
```

formula Φ, initializes the abstraction for every quantifier block of Φ, and then returns the result of the call to $\textsc{solve}_\exists$ or $\textsc{solve}_\forall$, shown in Algorithm 3 and 4, depending on the type of the leading quantifier block. The algorithm $\textsc{solve}_\mathcal{Q}(X, \Psi, \alpha_S)$ determines whether the quantified subformula $\mathcal{Q}X.\,\Psi$ is satisfiable under the condition that some clauses are already satisfied by assignments to variables bound at outer quantifiers (represented by α_S as discussed below).

The algorithms for quantified subformulas, $\textsc{solve}_\mathcal{Q}$, determine candidate assignments to the variables bound at that quantifier that meet the quantifier's objective (to satisfy and falsify the formula for $\exists$ and $\forall$ quantifier, respectively), or give a reason why there is no such assignment. If a quantifier is able to provide a candidate assignment, it is recursively verified by proceeding to the inner quantified subformula. A *conflict* occurs when the current assignment of variables definitely violates some clause (existential conflict) or satisfies all clauses (universal conflict). In case of such a conflict, the reason for this conflict is excluded at an outer quantifier level by refining the corresponding abstraction.

Abstraction $\boldsymbol{\theta}$. The formula θ represents, for every quantifier block, how the quantifier blocks's variables interact with the assignments of variables of other quantifier blocks. The algorithm guarantees that whenever a candidate assignment is generated, all variables bound at outer quantifier levels have a fixed assignment, and thus some (possibly empty) set of clauses is already satisfied. At an existential quantifier, the corresponding player then tries to satisfy more clauses with an assignment to the variables bound at this quantifier, while the universal player tries to find an assignment that make it harder to satisfy all clauses.

As in the case for 2QBF in the previous section, the interaction of abstractions is established by the clause satisfaction variables S with the same semantics as before, i.e., given some quantifier block $\mathcal{Q}X$ and assignment α_V of outer variables V (w.r.t. $\mathcal{Q}X$), for every clause $C_i \in \varphi$ the satisfaction variable $s_i \in S$ represents whether C_i is satisfied by α_V. This, however, is not enough for existential quantifiers as the existential player has the choice to either satisfy the clause or *assume* that the clause will be satisfied by an assignment of an inner quantifier. Thus, we add an additional set of variables A for every existential quantifier block $\exists X$, called *assumption variables*, with the intended semantics that variable a_i is set to false implies that the clause C_i is satisfied at this quantifier level (either by an assignment to X or an outer assignment α_V abstracted by α_S).

We are now going to define the abstraction that implements this intuition. Fix some quantifier block $\mathcal{Q}X$. To define the abstraction for $\mathcal{Q}X$, we split a clause C_i into three parts,

$$\begin{aligned}
C_i^< &\coloneqq \{l \in C_i \mid l \text{ bound before } \mathcal{Q}X\},\\
C_i^= &\coloneqq \{l \in C_i \mid \mathit{var}(l) \in X\}, \text{ and}\\
C_i^> &\coloneqq \{l \in C_i \mid l \text{ bound after } \mathcal{Q}X\}\ .
\end{aligned}$$

By definition, it holds that $C_i = C_i^< \dot\cup C_i^= \dot\cup C_i^>$.

For existential quantifiers, a clause C_i is encoded by a variable s_i that represents whether the clause C_i is *satisfied* by an assignment to variables outer to Y, the literals of the quantifiers's variables, and a variable a_i that indicates whether the clause is *assumed* to be satisfied by an inner assignment. For existential quantifier $\exists X$, the clausal abstraction for clause $C_i \in \varphi$ is defined as

$$\mathit{clabs}_{\exists X}(C_i) = \begin{cases} s_i \vee C_i^= & \text{if } \exists X \text{ is the innermost quantifier}\\ s_i \vee C_i^= \vee a_i & \text{otherwise}\end{cases} \tag{5}$$

During the execution of the algorithm, the algorithm potentially visits each quantifier multiple times to generate candidate assignments and assumptions. If those assumptions turn out to be wrong, that is, the corresponding assignment is losing for the existential player, the abstraction is refined. Such a refinement is a clause that contains only assumption variables A and represents sets of clauses that together cannot be satisfied by the inner quantifier.

The abstraction for universal quantifiers $\forall X$ is unchanged from the 2QBF algorithm, that is, we define

$$\mathit{clabs}_{\forall X}(C_i) = s_i \vee \neg C_i^= = \bigwedge_{l \in C_i^=} \bar{l} \vee s_i\ . \tag{6}$$

In contrast to existential quantifiers, universal quantifiers do not have separate sets of variables S and A; to define the abstraction we use only satisfaction variables S. This is merely a minor simplification that exploits the formula structure of universal quantifiers. The universal quantifier cannot make assumptions on the inner quantifiers: either a clause is falsified by some assignment to X or it is not. Refinements are represented as clauses over literals from variables S.

The clausal abstraction θ_X for some quantifier block $\mathcal{Q}X$ is defined as the conjunction over the abstractions of clauses

$$\theta_X \coloneqq \bigwedge_{C_i \in \varphi} \mathit{clabs}_{\mathcal{Q}X}(C_i)\ . \tag{7}$$

Algorithm for Existential Quantifiers. The algorithm $\textsc{solve}_\exists$ is shown in Algorithm 3. It decides whether the QBF $\exists X.\,\Phi$ is satisfiable under the assumption that the matrix φ is restricted according to the assignment α_S. The algorithm repeatedly generates candidate assignments by means of the abstraction θ_X (line 3). If the abstraction returns Unsat, there is no winning assignment for this quantifier, thus, the algorithm returns Unsat as well (line 10). Further, the reason for the negative result is given, represented by the assignment β_S, that indicates which clauses could not be satisfied simultaneously. If the abstraction returns Sat with assignment α we distinguish two cases. The first case is the base

Algorithm 3 Algorithm for existentially quantified formulas

1: **procedure** $\textsc{solve}_\exists(X,\, \Phi,\, \alpha_S)$
2:     **loop**
3:         **match** $\langle \textsc{sat}(\theta_X, \alpha_S),\ \ \Phi\rangle$ **as** ▷ assume satisfied and falsified clauses
4:             $\langle \mathsf{Sat}(\alpha),\ \ \forall Y.\, \Psi\rangle \Rightarrow$ ▷ $\Phi = \forall Y.\, \Psi$
5:                 $\alpha'_S \leftarrow \alpha_S \sqcup \{s_i \mapsto \mathbf{T} \mid a_i \in \alpha|^0_A\}$ ▷ update satisfied clauses
6:                 **match** $\textsc{solve}_\forall(Y,\, \Psi,\, \alpha'_S)$ **as** ▷ recursive verification
7:                     $\mathsf{Sat}(\beta_S) \Rightarrow$ **return** $\mathsf{Sat}(\beta_S \sqcap \alpha^+_S)$
8:                     $\mathsf{Unsat}(\beta_S) \Rightarrow\ \ \theta_X \leftarrow \theta_X \wedge \bigvee_{s_i \in \beta^0_S} \overline{a_i}$ ▷ refine θ_X
9:             $\langle \mathsf{Sat}(\_),\ \ \_\rangle \Rightarrow$ **return** $\mathsf{Sat}(\alpha^+_S)$ ▷ Φ is propositional
10:             $\langle \mathsf{Unsat}(\beta_S),\ \ \_\rangle \Rightarrow$ **return** $\mathsf{Unsat}(\beta_S)$
11:     **end loop**
12: **end procedure**

case of the recursion, that is, the inner formula is quantifier-free. The algorithm returns Sat together with the partial assignment α^+_S indicating which clauses have to be satisfied by outer quantifier such that the assignment α_X satisfies the matrix φ.

If the inner subformula is quantified, we split α into two parts $\alpha_A = \alpha|_A$ and $\alpha_X = \alpha|_X$. Then in line 5 we update α_S by marking those clauses as satisfied (set s_i to $\mathbf{T}$) that α_X satisfies and continue with the recursive verification using $\textsc{solve}_\forall$ (line 6) which, again, could either be Sat or Unsat. In the first case, the partial assignment β_S (line 7) indicates the clauses that are required to be satisfied. Before returning, we adapt this witness by the operation $\beta_S \sqcap \alpha^+_S$ in line 7 which removes those clauses that are already satisfied by α_X, i.e., clauses C_i where $\alpha_S(s_i) = \mathbf{F}$ and $\alpha_A(a_i) = \mathbf{F}$. In the second case where the verification is unsuccessful, the abstraction θ_X is refined by enforcing that some clause from the previously unsatisfied clauses is satisfied, before continuing with the next iteration.

Algorithm for Universal Quantifiers. The algorithm $\textsc{solve}_\forall$, shown in Algorithm 4, shares the same underlying concept and structure as $\textsc{solve}_\exists$ and differs only in minor

Algorithm 4 Algorithm for universally quantified formulas

procedure $\textsc{solve}_\forall(X,\, \Phi,\, \alpha_S)$
    loop
        match $\langle \textsc{sat}(\theta_X, \alpha^+_S),\ \ \Phi\rangle$ **as** ▷ assume satisfied clauses only
            $\langle \mathsf{Sat}(\alpha),\ \ \exists Y.\, \Psi\rangle \Rightarrow$ ▷ $\Phi = \exists Y.\, \Psi$
                match $\textsc{solve}_\exists(Y,\, \Psi,\, \alpha|_S)$ **as** ▷ recursive verification
                    $\mathsf{Unsat}(\beta_S) \Rightarrow$ **return** $\mathsf{Unsat}(\beta_S)$
                    $\mathsf{Sat}(\beta_S) \Rightarrow \theta_X \leftarrow \theta_X \wedge \bigvee_{s_i \in \beta^1_S} \overline{s_i}$ ▷ refine θ_X
            $\langle \mathsf{Unsat}(\beta_S),\ \ \_\rangle \Rightarrow$ **return** $\mathsf{Sat}(\beta_S)$
    end loop
end procedure

details that we discuss in the following. Due to the different abstractions, the algorithm only assumes already satisfied clauses (by assignments of outer variables), represented by α_S^+, when generating the candidate assignment in line 3. This also means that there is no need to update α_S after line 4, as $\alpha|_S$ already represents all satisfied clauses due to the definition of the universal abstraction. Further, the base case is missing as it is guaranteed that every universal quantifier is followed by an existential quantifier (otherwise it can be removed by universal reduction). The refinement in line 7 states that one of the previously satisfied clauses has to be falsified, starting with the next iteration.

Example 4. Consider again the formula given in Example 1:

$$\exists v, w. \forall x. \exists y, z. (w \vee x \vee y)(v \vee \overline{w})(x \vee \overline{y})(\overline{v} \vee z)(\overline{z} \vee \overline{x})$$

We build the abstractions

$$\begin{aligned}
\theta_{\{v,w\}} &= (s_1 \vee w \vee a_1)(s_2 \vee v \vee \overline{w} \vee a_2)(s_3 \vee a_3)(s_4 \vee \overline{v} \vee a_4)(s_5 \vee a_5), \\
\theta_{\{x\}} &= (s_1 \vee \overline{x})(s_3 \vee \overline{x})(s_5 \vee x), \text{ and} \\
\theta_{\{y,z\}} &= (s_1 \vee y)(s_2)(s_3 \vee \overline{y})(s_4 \vee z)(s_5 \vee \overline{z}) \ .
\end{aligned}$$

We give a possible execution of algorithm SOLVE. To improve readability, we use the propositional representation for assignments as cubes. Note that clause C_2 contains only variables of the outermost quantifier, thus, setting a_2 to true is a useless assumption. In Section 4.3 we discuss this (and other) improvements for the basic algorithm presented here, for now we just assume that the initial abstraction $\theta_{\{v,w\}}$ is $\theta_{\{v,w\}} \wedge \overline{a_2}$.

- $\text{SOLVE}_\exists(\{v, w\}, \forall x. \exists y, z. \varphi, \overline{s_1}\,\overline{s_2}\,\overline{s_3}\,\overline{s_4}\,\overline{s_5})$
- $\text{SAT}(\theta_{\{v,w\}}, \overline{s_1}\,\overline{s_2}\,\overline{s_3}\,\overline{s_4}\,\overline{s_5}) = \mathsf{Sat}(\overline{v}\,\overline{w}\,a_1\overline{a_2}a_3\overline{a_4}a_5)$
- $\alpha'_S = \overline{s_1}s_2\overline{s_3}s_4\overline{s_5}$
- $\text{SOLVE}_\forall(\{x\}, \exists y, z. \varphi, \alpha'_S)$
  - $\text{SAT}(\theta_{\{x\}}, s_2 s_4) = \mathsf{Sat}(x\, s_1 s_2 s_3 s_4 \overline{s_5})$
  - $\text{SOLVE}_\exists(\{y, z\}, \varphi, s_1 s_2 s_3 s_4 \overline{s_5})$
    - $\text{SAT}(\theta_{\{y,z\}}, s_1 s_2 s_3 s_4 \overline{s_5}) = \mathsf{Sat}(\overline{y}\,\overline{z})$
    - **return** $\mathsf{Sat}(s_1 s_2 s_3 s_4)$
  - $\theta'_{\{x\}} = \theta_{\{x\}} \wedge (\overline{s_1} \vee \overline{s_2} \vee \overline{s_3} \vee \overline{s_4})$
  - $\text{SAT}(\theta_{\{x\}}, s_2 s_4) = \mathsf{Sat}(\overline{x}\,\overline{s_1}s_2\overline{s_3}s_4 s_5)$
  - $\text{SOLVE}_\exists(\{y, z\}, \varphi, \overline{s_1}s_2\overline{s_3}s_4 s_5)$
    - $\text{SAT}(\theta_{\{y,z\}}, \overline{s_1}s_2\overline{s_3}s_4 s_5) = \mathsf{Unsat}(\overline{s_1}\,\overline{s_3})$
    - **return** $\mathsf{Unsat}(\overline{s_1}\,\overline{s_3})$
  - **return** $\mathsf{Unsat}(\overline{s_1}\,\overline{s_3})$
- $\theta'_{\{v,w\}} = \theta_{v,w} \wedge (\overline{a_1} \vee \overline{a_3})$

- $\textsc{sat}(\theta'_{\{v,w\}},\, \overline{s_1}\,\overline{s_2}\,\overline{s_3}\,\overline{s_4}\,\overline{s_5}) = \mathsf{Sat}(v\, w\, \overline{a_1}\, \overline{a_2} a_3 a_4 a_5)$
- $\alpha'_S = s_1 s_2 \overline{s_3}\,\overline{s_4}\,\overline{s_5}$
- $\textsc{solve}_\forall(\{x\},\, \exists y, z.\, \varphi,\, \alpha'_S)$
  - $\textsc{sat}(\theta_{\{x\}}, s_1 s_2) = \mathsf{Sat}(x\, s_1 s_2 s_3 \overline{s_4}\,\overline{s_5})$
  - $\textsc{solve}_\exists(\{y, z\},\, \varphi,\, s_1 s_2 s_3 \overline{s_4}\,\overline{s_5})$
    - $\textsc{sat}(\theta_{\{y,z\}},\, s_1 s_2 s_3 \overline{s_4}\,\overline{s_5}) = \mathsf{Unsat}(\overline{s_4}\,\overline{s_5})$
    - **return** $\mathsf{Unsat}(\overline{s_4}\,\overline{s_5})$
  - **return** $\mathsf{Unsat}(\overline{s_4}\,\overline{s_5})$
- $\theta''_{\{v,w\}} = \theta'_{v,w} \wedge (\overline{a_4} \vee \overline{a_5})$
- $\textsc{sat}(\theta''_{\{v,w\}},\, \overline{s_1}\,\overline{s_2}\,\overline{s_3}\,\overline{s_4}\,\overline{s_5}) = \mathsf{Unsat}(\overline{s_1}\,\overline{s_2}\,\overline{s_3}\,\overline{s_4}\,\overline{s_5})$
- **return** $\mathsf{Unsat}(\overline{s_1}\,\overline{s_2}\,\overline{s_3}\,\overline{s_4}\,\overline{s_5})$

4.2 Correctness

The proof of correctness generalizes the arguments made in Section 3.2 to formulas with arbitrary prefixes. Thus, the correctness argument presented in this section is an inductive argument over the quantifier prefix.

A substantial part of the formal arguments relies on the relation between the abstractions and the quantified Boolean formula that we formalize in the following. Let $\mathcal{Q}X$ and α_S be some quantifier and an assignment of satisfaction variables, respectively. In combination, we can interpret them as a new QBF that starts with the quantifier block $\mathcal{Q}X$, removes all literals that are bound prior to $\mathcal{Q}X$, and has only the clauses that are marked as unsatisfied by α_S. To formalize this intuition, we define an operator $\Phi|_{\alpha_S}^{\mathcal{Q}X}$ that restricts the matrix φ in a QBF Φ to those clauses $C_i \in \varphi$ such that $\alpha_S(s_i) = \mathbf{F}$ and removes all leading quantifiers up to $\mathcal{Q}X$. In detail, the resulting QBF has the same quantifier prefix starting with $\mathcal{Q}X$ and the matrix $\{C_i^{\geq} \mid C_i \in \varphi \wedge \alpha_S(s_i) = \mathbf{F}\}$ where $C_i^{\geq}$ refers to quantifier block $\mathcal{Q}X$. Note that variables bound by outer quantifiers are removed from the matrix. As an example, consider the formula $\Phi_{ex} = \exists v, w.\, \forall x.\, \exists y, z.\, (w \vee x \vee y)(v \vee \overline{w})(x \vee \overline{y})(\overline{v} \vee z)(\overline{z} \vee \overline{x})$ from the previous example: The formula $\Phi_{ex}|^{\forall x}_{\overline{s_1} s_2 \overline{s_3} s_4 \overline{s_5}}$ is equal to $\forall x.\, \exists y, z.\, (x \vee y)(x \vee \overline{y})(\overline{z} \vee \overline{x})$.

We start by stating simple properties about the abstractions after assuming some assignment α_S. Those are used in the induction proofs below.

Lemma 4. *Let Φ be a QBF with matrix φ and let α_S be an assignment over variables S.*

1. *Let $\exists X$ be the innermost quantifier block. It holds that $\theta_X[\alpha_S] = \bigwedge_{s_i \in \alpha_S^0} C_i^{=}$ which is equisatisfiable to $\Phi|_{\alpha_S}^{\exists X}$.*
2. *Let $\exists X$ be a (non-innermost) quantifier block of Φ. It holds that $\theta_X[\alpha_S] = \bigwedge_{s_i \in \alpha_S^0} (C_i^{=} \vee a_i)$.*
3. *Let $\forall X$ be a quantifier block of Φ. It holds that $\theta_X[\alpha_S^{+}] = \bigwedge_{s_i \in \alpha_S^0} (s_i \vee \neg C_i^{=})$.*

Proof. Follows immediately from the definition of the abstraction θ_X. $\square$

Let $\mathcal{Q}X.\,\overline{\mathcal{Q}}Y$ be a quantifier alternation of Φ. In the following proofs, we have to transform an assignment α_S of satisfaction variables (w.r.t. $\mathcal{Q}X$) to an assignment of satisfaction variables with respect to $\overline{\mathcal{Q}}Y$ by applying the effect of an assignment α_X to the variables X. Often, we will argue over the "optimal" assignment α_S^* of the satisfaction variables S in the abstraction θ_X to relate $\Phi|_{\alpha_S^*}^{\overline{\mathcal{Q}}Y}$ with $(\Phi|_{\alpha_S}^{\mathcal{Q}X})[\alpha_X]$. The following lemma states this connection formally.

Lemma 5. *Let $\mathcal{Q}X.\,\overline{\mathcal{Q}}Y$ be a quantifier alternation of Φ and let α_X and α_S be assignments as defined before. Further, let α_S^* be defined such that $\alpha_S^*(s_i) = \mathbf{T}$ if, and only if, $\alpha_S(s_i) = \mathbf{T}$ or $\alpha_X \vDash C_i|_X$. It holds that $(\Phi|_{\alpha_S}^{\mathcal{Q}X})[\alpha_X] = \Phi|_{\alpha_S^*}^{\overline{\mathcal{Q}}Y}$.*

Proof. The quantified formulas $(\Phi|_{\alpha_S}^{\mathcal{Q}X})[\alpha_X]$ and $\Phi|_{\alpha_S^*}^{\overline{\mathcal{Q}}Y}$ have the same prefix (both starting with $\overline{\mathcal{Q}}Y$) and the same matrix

$$\underbrace{\bigwedge_{\substack{C_i \in \varphi \\ \alpha_S(s_i) = \mathbf{F} \wedge \alpha_X \nvDash C_i|_X}} C_i^{>}}_{> \text{ w.r.t. } \mathcal{Q}X} = \underbrace{\bigwedge_{\substack{C_i \in \varphi \\ \alpha_S^*(s_i) = \mathbf{F}}} C_i^{\geq}}_{\geq \text{ w.r.t. } \overline{\mathcal{Q}}Y} . \qquad \square$$

We now have the necessary preconditions to state the inductive arguments formally. The following lemma states that $\textsc{solve}_{\mathcal{Q}}$ returns Sat if the given QBF is satisfiable. Further, the returned witness represents the necessary condition for satisfiability in form of a partial assignment β_S. Recall that for some partial assignment β, the notation $\beta[\bot \mapsto b]$ describes the complete assignment where undefined values are set to $b \in \mathbb{B}$.

Lemma 6. *Let $\mathcal{Q}X.\,\Psi$ be a quantified subformula of a QBF Φ with matrix φ and let α_S be an assignment of variables S. If $\Phi|_{\alpha_S}^{\mathcal{Q}X}$ is true $\textsc{solve}_{\mathcal{Q}}(X,\,\Psi,\,\alpha_S)$ returns $\mathsf{Sat}(\beta_S)$ where $\beta_S \sqsubseteq \alpha_S^+$ and $\Phi|_{\beta_S[\bot\mapsto\mathbf{F}]}^{\mathcal{Q}X}$ is true.*

Proof. We prove the statement by structural induction over the quantifier prefix. The base case follows immediately by Lemma 4.1. For the induction step, we consider existential and universal quantification separately. For existential quantifier $\exists X$, there has to be a satisfying assignment α_X by the QBF semantics and we show that this assignment is a satisfying assignment for the abstraction θ_X. Together with the optimal set of assumptions, we can use the induction hypothesis to build a witnessing partial assignment. Completeness follows from the fact that there are only finitely many different refinement clauses and the property that assignment α_X cannot be excluded by some refinement. For universal quantifier $\forall X$, every assignment α_X is satisfying, thus, we show that every satisfying assignment of the abstraction leads to a subsequent refinement. Thus, the abstraction becomes unsatisfiable (under the given assumption α_S) eventually, and the algorithm returns Sat with a witness satisfying the requirement. The detailed proof follows.

Induction Base. Let $\exists X.\,\varphi$ be the innermost quantifier of Φ and let α_S be such that $\Phi|_{\alpha_S}^{\exists X}$ is true. By Lemma 4.1, the truth of $\Phi|_{\alpha_S}^{\exists X}$ witnesses the satisfiability of $\theta_X[\alpha_S]$. Further, the algorithm $\textsc{solve}_{\exists}$ returns $\mathsf{Sat}(\alpha_S^+)$ (line 9) and $\alpha_S^+[\bot \mapsto \mathbf{F}]$ is equivalent to α_S.

Induction Step ($\mathcal{Q} = \exists$). Let $\exists X. \forall Y$ be an arbitrary quantifier alternation of Φ and let α_S be such that $\Phi|^{\exists X}_{\alpha_S}$ is true. By Lemma 4.2 it holds that

$$\theta_X[\alpha_S] = \bigwedge_{s_i \in \alpha_S^0} (C_i^= \vee a_i) \quad .$$

Since $\Phi|^{\exists X}_{\alpha_S}$ is true, there is a satisfying assignment α_X for the variables X such that $(\Phi|^{\exists X}_{\alpha_S})[\alpha_X]$ (a QBF starting with quantifier $\forall Y$) is true. Define α_A^* as $\alpha_A^*(a_i) = \mathbf{F}$ if, and only if, $\alpha_X \vDash C_i^=$. Thus, α_A^* is the assignment with the smallest number of assumptions ($\alpha_A^*(a_i) = \mathbf{T}$) for the given assignment α_X. The combined assignment $\alpha_X \dot\sqcup \alpha_A^*$ is a satisfying assignment of the initial abstraction $\theta_X[\alpha_S]$ by construction. We perform a case distinction on the returned assignment of the SAT solver in line 3.

- We assume that the SAT call in line 3 returns $\alpha_X \dot\sqcup \alpha_A^*$. Let α_S^* be the assignment constructed from α_S and α_A^* in line 5. By Lemma 5, it holds that $(\Phi|^{\exists X}_{\alpha_S})[\alpha_X] = \Phi|^{\forall Y}_{\alpha_S^*}$ is true. By induction hypothesis we deduce that $\text{SOLVE}_\forall$ returns $\mathsf{Sat}(\beta_S)$ where $\Phi|^{\forall Y}_{\beta_S[\bot \mapsto 0]}$ is true. Subsequently, $\text{SOLVE}_\exists$ returns $\mathsf{Sat}(\beta'_S)$ (line 7), where $\beta'_S = \beta_S \sqcap \alpha_S^+$.

  As the algorithm returns $\mathsf{Sat}(\beta'_S)$, it remains to show that $\Phi|^{\exists X}_{\beta'_S[\bot \mapsto \mathbf{F}]}$ is true. For every clause that is removed from β_S by the intersection with α_S^+, it holds that this clause is satisfied by the assignment α_X: Assume $s_i \in S$ is removed by the intersection, that is, $\beta_S(s_i) = \mathbf{T}$ and $\alpha_S(s_i) = \mathbf{F}$. We know that $\beta_S \sqsubseteq {\alpha_S^*}^+ = (\alpha_S \sqcup \{s_i \mapsto 1 \mid a_i \in {\alpha_A^*}^0\})^+$ by induction hypothesis and the construction of α_S^* in line 5. Hence, $\alpha_A^*(a_i) = \mathbf{F}$ and together with $\alpha_S(s_i) = \mathbf{F}$ we conclude that $\alpha_X \vDash C_i^=$ due to the definition of $\mathit{clabs}_{\exists X}$ in Equation 5.

- Assume that the SAT call in line 3 returns an assumptio α'_A different to α_A^*. Either α'_A corresponds to α_X and is non-minimal, i.e., ${\alpha_A^*}^+ \sqsubseteq {\alpha'_A}^+$, or it corresponds to a different assignment α'_X. The call to $\text{SOLVE}_\forall$ may either return Sat or a counterexample $\mathsf{Unsat}(\beta_S)$. We consider the latter case as in the former case $\text{SOLVE}_\exists$ also returns Sat and the same argumentation as in the previous case applies.

  The subsequent refinement in line 8 requires that one of the unsatisfied clauses C_i with $\beta_S(s_i) = \mathbf{F}$ has to be satisfied in the next iteration and the corresponding refinement clause is $\psi \coloneqq \bigvee_{s_i \in \beta_S^0} \overline{a}_i$. By construction of α_A^* as the optimal assignment corresponding to α_X, $\alpha_A^* \nvDash \psi$ contradicts that α_X is a satisfying assignment of $\Phi|^{\exists X}_{\alpha_S}$. Hence, $\alpha_X \dot\sqcup \alpha_A^*$ is still a satisfying assignment for the refined abstraction $\theta'_X[\alpha_S]$. The refinement also reduces the number of A assignments by at least 1 and, thus, brings us one step closer to termination.

Induction Step ($\mathcal{Q} = \forall$). Let $\forall X. \exists Y$ be a quantifier alternation of Φ and let α_S be such that $\Phi|^{\forall X}_{\alpha_S}$ is true. For every assignment α_X, it holds that $(\Phi|^{\forall X}_{\alpha_S})[\alpha_X]$ (a QBF starting with quantifier $\exists Y$) is true. By Lemma 4.3 it holds that

$$\theta_X[\alpha_S^+] = \bigwedge_{s_i \in \alpha_S^0} (s_i \vee \neg C_i^=) \quad .$$

Thus, in order to set s_i to false for some i, every literal $l \in C_i^=$ has to be assigned negatively. Fix some arbitrary assignment α_X. Let α_S^* be the assignment with $\alpha_S^*(s_i) = \mathbf{T}$ if, and only if, $\alpha_S(s_i) = \mathbf{T}$ or $\alpha_X \vDash C_i^{=}$. Note that α_S^* is minimal with respect to the number of positively assigned s_i corresponding to α_X. For every α'_S returned from the SAT solver in line 3 (assuming α_X is fixed) it holds that ${\alpha_S^*}^+ \sqsubseteq {\alpha'_S}^+$ by the minimality of α_S^*. By Lemma 5, it holds that $(\Phi|^{\forall X}_{\alpha_S})[\alpha_X] = \Phi|^{\exists Y}_{\alpha_S^*}$ is true and thereby $\Phi|^{\exists Y}_{\alpha'_S}$ is true as its matrix contains a subset of the clauses of $\Phi|^{\exists Y}_{\alpha_S^*}$. By induction hypothesis we deduce that $\textsc{solve}_\exists$ returns $\mathsf{Sat}(\beta'_S)$ where $\beta'_S \sqsubseteq \alpha'_S$ and $\Phi|^{\exists Y}_{\beta'_s[\bot \mapsto 0]}$ is true. The subsequent refinement in line 7 reduces the number of S assignments, so the abstraction θ_X becomes unsatisfiable (under the assumption α_S) eventually and the loop terminates with $\mathsf{Sat}(\beta_S)$ in line 8. Let θ'_X be the abstraction after the termination of the loop. $\beta_S \sqsubseteq \alpha_S^+$ holds as β_S are the failed assumptions of the SAT call $\textsc{sat}(\theta'_X, \alpha_S^+)$.

It remains to show that $\Phi|^{\forall X}_{\beta_S[\bot \mapsto \mathbf{F}]}$ is true. Assume for contradiction that there is some α_X such that $(\Phi|^{\forall X}_{\beta_S[\bot \mapsto \mathbf{F}]})[\alpha_X]$ is false. We know that $\theta'_X[\alpha_X \dot\sqcup \beta_S]$ is unsatisfiable. Either the initial abstraction $\theta_X[\alpha_X \dot\sqcup \beta_S]$ was unsatisfiable, which leads to a contradiction due to Lemma 5, or the assignment α_X was excluded due to refinements. As the refinement only excludes S assignments β''_S such that $\Phi|^{\exists Y}_{\beta''_S[\bot \mapsto \mathbf{F}]}$ is true, this leads to a contradiction as well. □

The following lemma states the reverse direction, that the algorithm terminates with the correct result on false formulas. The arguments used in the proof are very similar to the one for true formulas, but the differences are enough to justify their inclusion.

Lemma 7. *Let $\mathcal{Q}X.\,\Psi$ be a quantified subformula of a QBF Φ with matrix φ and let α_S be an assignment of variables S. If $\Phi|^{\mathcal{Q}X}_{\alpha_S}$ is false $\textsc{solve}_{\mathcal{Q}}(X, \Psi, \alpha_S)$ returns $\mathsf{Unsat}(\beta_S)$ where $\beta_S \sqsubseteq \alpha_S^-$ and $\Phi|^{\mathcal{Q}X}_{\beta_S[\bot \mapsto \mathbf{T}]}$ is false.*

Proof. The structure of the proof is similar to the proof of Lemma 6, that is, a structural induction over the quantifier prefix. For existential quantifier $\exists X$, every assignment α_X leads to a false QBF. We can use the induction hypothesis for every assignment produced by the abstraction θ_X as the abstraction computes an under-approximation of the satisfied clauses with respect to α_X. We show that the subsequent refinement excludes at least the given assignment, thus, the abstraction becomes unsatisfiable eventually (under the given assumption α_S). It remains to show that the returned partial assignment satisfies is a witness for the falsity of the subformula. For universal quantifier $\forall X$, there is some assignment α_X that leads to a false QBF. We show that the algorithm eventually reaches this assignment (or another assignment that leads to unsatisfiability). Applying induction hypothesis leads to a witnessing partial assignment. The detailed proof follows.

Induction Base. Let $\exists X.\,\varphi$ be the innermost quantifier of Φ and let α_S be such that $\Phi|^{\exists X}_{\alpha_S}$ is false. By Lemma 4.1, $\theta_X[\alpha_S]$ is unsatisfiable. Let β'_S be the failed assumptions from the call to $\textsc{sat}(\theta_X, \alpha_S)$, i.e., $\beta'_S \sqsubseteq \alpha_S^-$ and $\theta_X[\beta'_S]$ is unsatisfiable. Again by Lemma 4.1 it holds that $\Phi|^{\exists X}_{\beta'_S[\bot \mapsto \mathbf{T}]}$ is false which concludes the induction base as $\mathsf{Unsat}(\beta'_S)$ is returned from $\textsc{solve}_\exists$.

Induction Step ($\mathcal{Q} = \exists$). Let $\exists X.\,\forall Y$ be a quantifier alternation of Φ and let α_S be such that $\Phi|^{\exists X}_{\alpha_S}$ is false. For every assignment α_X, it holds that $(\Phi|^{\exists X}_{\alpha_S})[\alpha_X]$ is false. By Lemma 4.2

it holds that

$$\theta_X[\alpha_S] = \bigwedge_{s_i \in \alpha_S^0} (C_i^{=} \vee a_i) \quad .$$

The abstraction θ_X is initially satisfiable for every choice of α_S (every a_i can be set to true)[3]. Let α be such a satisfying assignment of $\theta_X[\alpha_S]$. We define $\alpha_X \coloneqq \alpha|_X$ and $\alpha_A \coloneqq \alpha|_A$. By Lemma 4.2, $\alpha_X \nvDash C_i^{=}$ implies that $\alpha_A(a_i) = \mathbf{T}$. We define the assignment with optimal assumptions α_A^* as $\alpha_A^*(a_i) = \mathbf{F}$ if, and only if, $\alpha_X \vDash C_i^{=}$. Note that $\alpha_X \dot\sqcup \alpha_A^*$ is a satisfying assignment of $\theta_X[\alpha_S]$. We show that even with optimal assumptions α_A^*, the quantified subformula is unsatisfiable and the subsequent refinement step excludes at least assignment $\alpha_S \dot\sqcup \alpha_A$ from the abstraction θ_X.

Let α'_S and α^*_S be the assignments after line 5 with respect to α_A and α_A^*, respectively. From the construction, we know that ${\alpha_A}^- \sqsubseteq {\alpha_A^*}^-$, by the optimality of α_A^*, and thereby ${\alpha'_S}^+ \sqsubseteq {\alpha^*_S}^+$. We deduce that $\Phi|^{\forall Y}_{\alpha'_S}$ is false, as the clauses in the matrix $\Phi|^{\forall Y}_{\alpha'_S}$ are a superset of those in the matrix of $\Phi|^{\forall Y}_{\alpha^*_S}$ which is equal to $(\Phi|^{\exists X}_{\alpha_S})[\alpha_X]$ by Lemma 5. By induction hypothesis, $\textsc{solve}_\forall$ with assignment α'_S returns $\mathsf{Unsat}(\beta_S)$ such that $\beta_S \sqsubseteq {\alpha'_S}^-$ and $\Phi|^{\forall Y}_{\beta_S[\bot \mapsto \mathbf{T}]}$ is false. As $\beta^0_S \subseteq {\alpha'_S}^0 = \{s_i \in S \mid \alpha_S(s_i) = \mathbf{F} \wedge \alpha_A(a_i) = \mathbf{T}\}$, the following refinement with clause $\bigvee_{s_i \in \beta^0_S} \overline{a_i}$ excludes assignment $\alpha_S \dot\sqcup \alpha_A$ from θ_X. As there are only finitely many refinement clauses, the SAT call in line 3 eventually becomes unsatisfiable when assuming α_S. Let θ'_X be the abstraction at this point and let β'_S be the failed assumptions, i.e., $\beta'_S \sqsubseteq \alpha^-_S$.

Let $\alpha''_S = \beta'_S[\bot \mapsto \mathbf{T}]$. It remains to show that $\Phi|^{\exists X}_{\alpha''_S}$ is false. Assume for contradiction that there is some α_X such that $(\Phi|^{\exists X}_{\alpha''_S})[\alpha_X]$ is true. It holds that $\theta'_X[\alpha_X \dot\sqcup \alpha''_S]$ is unsatisfiable, whereas initially, $\theta_X[\alpha_X \dot\sqcup \alpha''_S]$ is satisfiable. Thus, the assignment α_X was excluded due to refinements. As the refinement only excludes assignments corresponding to some S assignment β''_S such that $\Phi|^{\forall Y}_{\beta''_S[\bot \mapsto \mathbf{T}]}$ is false, this contradicts our assumption.

Induction Step ($\mathcal{Q} = \forall$). Let $\forall X. \exists Y$ be a quantifier alternation of Φ and let α_S be such that $\Phi|^{\forall X}_{\alpha_S}$ is false, that is, there is an assignment α_X such that $(\Phi|^{\forall X}_{\alpha_S})[\alpha_X]$ is false. By Lemma 4.3 it holds that

$$\theta_X[\alpha^+_S] = \bigwedge_{s_i \in \alpha_S^0} (s_i \vee \neg C_i^{=}) \quad .$$

$\theta_X[\alpha^+_S]$ is initially satisfiable. Let α be a satisfying assignment of $\theta_X[\alpha^+_S]$ and define $\alpha'_X \coloneqq \alpha|_X$ and $\alpha'_S = \alpha|_S$. Given α_X from above, we define the optimal corresponding assignment α^*_S as $\alpha^*_S(s_i) = \mathbf{T}$ if, and only if, $\alpha_S(s_i) = \mathbf{T}$ or $\alpha_X \vDash C_i^{=}$. Note that α_S and α^*_S correspond to quantifier $\forall X$ and $\exists Y$, respectively. If $\alpha'_S = \alpha^*_S$, the call to $\textsc{solve}_\exists$ returns $\mathsf{Unsat}(\beta_S)$ where $\beta_S \sqsubseteq {\alpha^*_S}^-$ and $\Phi|^{\exists Y}_{\beta_S[\bot \mapsto \mathbf{T}]}$ is false by induction hypothesis as $(\Phi|^{\forall X}_{\alpha_S})[\alpha_X] = \Phi|^{\exists Y}_{\alpha^*_S}$ (Lemma 5) is false. Subsequently, $\textsc{solve}_\forall$ returns $\mathsf{Unsat}(\beta_S)$ (line 6). $\beta_S \sqsubseteq \alpha^-_S$ follows from ${\alpha^*_S}^- \sqsubseteq \alpha^-_S$ due to the monotonicity of the abstraction: if $\alpha^*_S(s_i) = \mathbf{F}$, then $\alpha_S(s_i) = \mathbf{F}$.

Let $\alpha'_S \neq \alpha^*_S$ and assume that $\textsc{solve}_\exists$ returns $\mathsf{Sat}(\beta_S)$. Subsequently, θ_X is refined by adding the the clause $\psi \coloneqq \bigvee_{s \in \beta^1_S} \overline{s_i}$. Assume for contradiction that $\alpha^*_S \nvDash \psi$, i.e., that α^*_S is excluded by the refinement. Remember that α^*_S was constructed as the optimal assignment

3. In Section 4.3 we describe improvements of the abstraction.

corresponding to α_X. Hence, the exclusion contradicts that α_X is a witness that $\Phi|^{\forall X}_{\alpha_S}$ is false. Thus, $\alpha_X \dot{\sqcup} \alpha^*_S$ remains a satisfying assignment of the refined abstraction. The refinement reduced the number of S assignments and, thus, some falsifying assignment α_X is reached eventually. □

Since the main algorithm SOLVE directly calls into SOLVE$_{\mathcal{Q}}$, the following theorem follows immediately from Lemma 6 and 7.

Theorem 3. SOLVE *returns* Sat *if, and only if,* Φ *is true.*

4.3 Optimizations

In this section, we introduce optimizations for the basic algorithm presented in Section 4.1. We start with two optimizations already described in the initial paper describing clausal abstraction [66]. We then proceed to improvements of the abstraction followed by algorithmic improvements. Some of these optimizations are generalized from the 2QBF fragment in Section 3.3.

Stronger Refinements. An existential conflict for quantifier alternation $\exists X.\, \forall Y$ of QBF Φ is a partial assignment β_S such that $\Phi|^{\forall Y}_{\beta_S[\bot \mapsto \mathbf{T}]}$ is false. Intuitively, β_S represents a set of clauses $\mathcal{C} = \{C_i \mid s_i \in \beta^0_S\}$ that could not be satisfied by the inner quantifier, i.e., replacing the matrix of Φ by $\mathcal{C}^> = \{C^>_i \mid s_i \in \beta^0_S\}$ results in a false QBF (Lemma 7). Refinements for such a partial assignment (line 8 of Algorithm 3), thus, assert that one of these clauses has to be satisfied at quantifier $\exists X$ to prevent this situation.

In certain cases, we can strengthen the refinement by excluding a conjunction of "equivalent" clauses, that are clauses that can replace the original clause and would let to the same result. Let $\mathcal{C}$ be the representation of some existential conflict, let $C_i \in \mathcal{C}$ and let $\mathcal{C}'$ be $\mathcal{C} \setminus C_i$. If there is some $C_j \in \varphi$, such that $C^>_j \subseteq C^>_i$, then $\mathcal{C}' \cup C_j$ is an existential conflict as well. Thus, we change the refinement to exclude all equivalent existential conflicts by modifying it to

$$\bigvee_{s_i \in \beta^0_S} \bigwedge_{\substack{C_j \in \varphi \\ C^>_j \subseteq C^>_i}} \overline{a_j} \ . \tag{8}$$

In [72], we have shown that this improved refinement makes the underlying proof system exponentially more succinct.

Tree-shaped Quantifier Prefix. As a preprocessing, we apply the well known mini-scoping rule

$$\forall X.\, \exists Y \exists Z.\, \varphi(X,Y) \wedge \psi(X,Z) \equiv (\forall X.\, \exists Y.\, \varphi(X,Y)) \wedge (\forall X.\, \exists Z.\, \psi(X,Z)),$$

that is, at every existential quantifier block we search for a partitioning of the matrix into independent formulas. By applying this rule bottom-up, we get a tree-shaped quantifier prefix. Note, that this tree only branches after an existential quantifier, hence, we modify the algorithm to split the current entry according to the partitioning and solve every child individually. This can be used to solve independent branches in parallel [71].

Abstraction Improvements. We describe improvements to the way the abstractions are built, that is, reducing the number of satisfaction and assumption variables. These optimizations are similar to the ones described in Section 3.3. Fix some QBF Φ. Let $\exists X.\, \Psi$ be a quantified subformula of Φ and let C_i some clause. If $C_i^<$ is empty, i.e., the clause contains no variable bound at some outer quantifier, then the assumption variable s_i at this quantifier can be always assumed to be false. Further, if $C_i^>$ is empty, then a_i can be assumed to be false and, thus, be removed. This requires a change to Algorithm 3, though: in the return $\mathsf{Sat}(\beta_S \sqcap \alpha_S^+)$ in line 7 we have to add those clauses without assumption variable that are not satisfied by the current assignment, i.e., it has to change to $\mathsf{Sat}((\beta_S \sqcup \{s_i \mapsto \mathbf{T} \mid C_i^> = \emptyset \land \alpha_X \nvDash C_i^=\}) \sqcap \alpha_S^+)$. Independent of the quantifier type, it is possible to omit building the abstraction for clauses with $C_i^= = \emptyset$ where the given quantifier has no influence on the satisfaction of the clause. Especially, we do not need to add the satisfaction and assumption variables initially. This is possible, since the updates to the satisfaction assignment α_S are monotone: if a clause is satisfied at some outer quantifier, it is guaranteed to be satisfied by every inner quantifier (see line 5 of Algorithm 3 and lines 3–4 of Algorithm 4). However, we may need to add them during solving in case there is some refinement involving those variables.

We generalize the simplifications for the universal abstraction introduced for 2QBF in Section 3.3:

- If some clause $C_i \in \varphi$ is a universal unit clause, i.e., $C|_X = \{l\}$ for some literal l with $var(l) \in X$, and there are no outer variables ($C_i^< = \emptyset$) then the shared variable s_i can be replaced by the negation $\bar{l}$ of the literal.
- If there is a pair of clauses $C_i, C_j \in \varphi$ with $i \neq j$ such that those clauses are equal with respect to the variables bound at this quantifier, i.e., $C_i^\leq = C_j^\leq$, then the same shared variable s_i can be used for both clauses.

Algorithmic Improvements. We recap generalizations of the algorithmic improvements described for the 2QBF algorithm in Section 3.3. Given some assignment α_X from the abstraction, we construct the corresponding "optimal" assignment of α_A (Algorithm 3) and α_S (Algorithm 4) as described by Lemma 5, respectively. For the propositional case of existential quantifier $\exists X$, the same optimizations as discussed in Section 3.3 can be applied: We set $\alpha_S(s_i) = \mathbf{F}$ before line 9 if $\alpha_S(s_i) = \mathbf{T}$ and $\alpha_X \vDash C_i^=$. Further, we may change the assignment α_X if such a change satisfies strictly more clauses.

We also generalize the optimization of refinement clauses due to subsumed literals described in Section 3.3. Given a partial assignment β_S representing a conflict in line 8 of $\textsc{solve}_\exists$ (Algorithm 3). If there are two clauses C_i and C_j with $C_i^\leq \subseteq C_j^\leq$ and $\beta_S(s_i) = \beta_S(s_j) = \mathbf{F}$, then we can set $\beta_S(s_i) = \bot$, which removes $\overline{a_i}$ from the refinement clause. Given a partial assignment β_S representing a conflict in line 7 of $\textsc{solve}_\forall$ (Algorithm 4). If there are two clauses C_i and C_j with $C_i^\leq \subseteq C_j^\leq$ and $\beta_S(s_i) = \beta_S(s_j) = \mathbf{T}$, then we can set $\beta_S(s_j) = \bot$, which removes $\overline{s_j}$ from the refinement clause.

The presented algorithms refine conflicts at the earliest point possible, e.g., if a universal quantifier returns $\mathsf{Unsat}(\beta_S)$ (line 8 of Algorithm 3), the abstraction at the existential quantifier is refined immediately. In some cases, this refinement is not needed as the existential quantifier does not control any of the refined clauses, that is, for all $C_i \in \varphi$ with $s_i \in \beta_S^0$

it holds that $C_i^{=} = \emptyset$. The following SAT call in line 3 is unsatisfiable and β_S is a possible failed assumption. Thus, the conflict is just propagated. As an example, consider the prefix $\exists x \forall v \exists y \forall w \exists z$ and a clauses $(x \vee \overline{v} \vee w \vee \overline{z})(x \vee \overline{v} \vee w \vee z)$. Given the assignment $\overline{x}v\overline{w}$, the quantifier $\exists z$ cannot satisfy both clauses simultaneously. The refinement at quantifier $\exists y$ produces the same conflict again as y has no impact. We add a check to Algorithm 3 and Algorithm 4 whether a conflict β_S can be propagated, thus, saving the cost of the refinement and the subsequent SAT call. This optimization was first described as part of the clause selection algorithm [46].

5. Function Extraction

For quantified Boolean formulas, the solving result goes beyond the binary decision problem discussed in the previous sections. Especially when using QBF as a target for applications, the witnessing Boolean functions are of great importance. Using Skolem functions, one can directly construct realizing implementations for synthesis problems encoded to QBF [15,16, 23,24]. And even in the negative case, the Herbrand functions may give valuable information about the underlying reason [36]. Another benefit of function extraction is the *certification* of the solving result, i.e., having a verifiable witness for the solving result. In this section, we present the function extraction approach for the clausal abstraction algorithm.

The function extraction is based on the correctness proof given in Section 4.2. Given a QBF Φ, some quantifier block $\mathcal{Q}X$ of Φ, and some assignment of satisfaction variables α_S. Lemma 6 shows that there is an assignment to α_X such that the subformula $(\Phi|_{\alpha_S}^{\exists X})[\alpha_X]$ is true if $\Phi|_{\alpha_S}^{\exists X}$ is true. Dually, Lemma 7 states that an assignment to α_X exists such that the subformula $(\Phi|_{\alpha_S}^{\forall X})[\alpha_X]$ is false if $\Phi|_{\alpha_S}^{\forall X}$ is false. Thus, the function extraction amounts to logging the relevant results during the execution of the algorithm, that is after the successful verification of the candidate assignment. In the following, we determine the relevant information that is needed for the extraction, the data structure in which the information is stored, and an extraction algorithm that returns the Skolem and Herbrand functions, respectively.

Recursion Tree. The execution of the clausal abstraction algorithm can be represented as a tree, where the nodes represent quantifiers $\mathcal{Q}X$ and the edges determines the truth value and witnessing assignments α_X. Formally, a node in the recursion tree is a pair $\langle \mathcal{Q}X, \alpha_S \rangle$ and there is an edge from $\langle \mathcal{Q}X, \alpha_S \rangle$ to $\langle \overline{\mathcal{Q}}Y, \alpha'_S \rangle$ labeled with the candidate assignment α_X and the result $res(\beta_S)$ returned from $\textsc{solve}_{\mathcal{Q}}$ if, and only if, (1) $\overline{\mathcal{Q}}Y$ is the quantifier block following $\mathcal{Q}X$, (2) $(\Phi|_{\alpha_S}^{\mathcal{Q}X})[\alpha_X] =^{4.} \Phi|_{\alpha'_S}^{\overline{\mathcal{Q}}Y}$, and (3) res is the result of $\Phi|_{\alpha'_S}^{\overline{\mathcal{Q}}Y}$ where $\Phi|_{\beta_S[\perp \mapsto \mathbf{F}]}^{\overline{\mathcal{Q}}Y}$ is true if $res = \mathsf{Sat}$ and $\Phi|_{\beta_S[\perp \mapsto \mathbf{T}]}^{\overline{\mathcal{Q}}Y}$ is false otherwise. The leaf nodes $\langle \exists X, \alpha_S \rangle$ are labeled with the result of the propositional formula $\Phi|_{\alpha_S}^{\exists X}$, that is, either Unsat or $\mathsf{Sat}(\alpha_X)$. The root node for some formula $\Phi = \mathcal{Q}X. \Psi$ is the designated node $\langle \mathcal{Q}X, \{s_i \mapsto \mathbf{F} \mid C_i \in \varphi\} \rangle$. We depict such a recursion tree in Figure 1.

After the algorithm terminates, we use the recursion tree to extract the relevant information to build Skolem and Herbrand functions, respectively. Note that for true QBFs and existential nodes as well as false QBFs and universal nodes, the respective nodes have

4. The equality holds if we assume optimal assumptions w.r.t. α_X as discussed in Section 4.3 about algorithmic improvements.

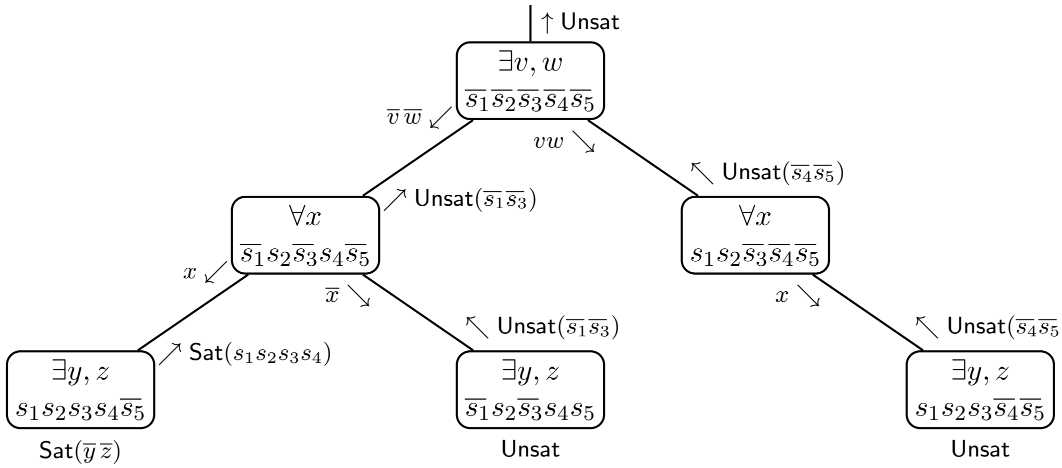


Figure 1: Recursion tree corresponding to the execution of $\textsc{solve}_\mathcal{Q}$ on the formula $\exists v, w.\, \forall x.\, \exists y, z.\, (w \lor x \lor y)(v \lor \overline{w})(x \lor \overline{y})(\overline{v} \lor z)(\overline{z} \lor \overline{x})$ as shown in Example 4.

exactly one outgoing edge where the candidate assignment was verified recursively. Due to the correctness lemmata Lemma 6 and Lemma 7, only the labeling of the edges, i.e., the assignment α_X and the returned partial assignment β_S are relevant. Thus, we store a list of these *verified candidates* as a sequence of pairs $\langle \beta_S, \alpha_X \rangle \in (\mathcal{A}_\perp(S) \times \mathcal{A}(X))$ for every quantifier block $\mathcal{Q}X$.

Function Extraction. We define a function $inv_{\mathcal{Q}X} \colon \mathcal{A}_\perp(S) \to \mathcal{B}(V)$ which, for a given quantifier block $\mathcal{Q}X$, maps an assignment β_S to a propositional formula over variables V bound by outer quantifiers (with respect to $\mathcal{Q}X$). Intuitively, $inv_{\mathcal{Q}X}(\beta_S)$ describes those assignments that lead to β_S in the abstraction of quantifier block $\mathcal{Q}X$. We define $inv_{\mathcal{Q}X}$ as

$$inv_{\mathcal{Q}X}(\beta_S) \coloneqq \begin{cases} \bigwedge_{s_i \in \beta_S^1} C_i^< & \text{if } \mathcal{Q} = \exists \\ \bigwedge_{s_i \in \beta_S^0} \neg C_i^< & \text{otherwise} \end{cases} . \tag{9}$$

Let $\langle \beta_S^1, \alpha_X^1 \rangle \ldots \langle \beta_S^n, \alpha_X^n \rangle$ be the pairs of verified candidates corresponding to quantifier block $\mathcal{Q}X$ and let $x \in X$ be some variable, the function $f_x \colon \mathcal{A}(V) \to \mathbb{B}$ is defined as

$$f_x \coloneqq \bigvee_{i=1}^{n} \left((\alpha_X^i(x) = 1) \land inv_{\mathcal{Q}X}(\beta_S^i) \land \bigwedge_{j<i} \neg inv_{\mathcal{Q}X}(\beta_S^j) \right) . \tag{10}$$

The definition of $inv_{\mathcal{Q}X}$ allows that f_x may depend on all variables bound at outer quantifiers, even those that are of the same quantifier type. By replacing those variables with their extracted functions, one can make sure that f_x depends only on its dependencies $dep(x)$. The size of f_x, measured in terms of distinct subformulas, is linear in the number of pairs. The function $f_X \colon \mathcal{A}(V) \to \mathcal{A}(X)$ is defined as the union over all f_x for $x \in X$, formally $f_X(\alpha_V) \coloneqq \bigsqcup_{x \in X} \{x \mapsto f_x(\alpha_V)\}$. The Skolem and Herband function are then defined as the union over the functions f_X for every $\mathcal{Q}X$ where $\mathcal{Q} = \exists$ for Skolem functions and $\mathcal{Q} = \forall$ for Herbrand functions.

Example 5. We show the function extraction for our running example $\exists v, w. \forall x. \exists y, z. (w \vee x \vee y)(v \vee \overline{w})(x \vee \overline{y})(\overline{v} \vee z)(\overline{z} \vee \overline{x})$. From the recursion tree in Figure 1, we extract the sequence $\langle \overline{s_1}\overline{s_3}, \overline{x} \rangle \langle \overline{s_4}\overline{s_5}, x \rangle$ as described above. Applying the definition of $inv_{\forall x}$, we get

$$\begin{aligned} inv_{\forall x}(\overline{s_1}\overline{s_3}) &= \neg C_1^< \wedge \neg C_3^< = \neg w \text{ and} \\ inv_{\forall x}(\overline{s_4}\overline{s_5}) &= \neg C_4^< \wedge \neg C_5^< = v \ . \end{aligned}$$

Thus, the Herbrand function f_x is defined as

$$f_x(v, w) = inv_{\forall x}(\overline{s_4}\overline{s_5}) \wedge \neg inv_{\forall x}(\overline{s_1}\overline{s_3}) = v \wedge w \ .$$

f_x depends solely on its dependencies and is functionally correct as $\varphi[f_x]$ is equal to

$$\begin{aligned} &(w \vee (v \wedge w) \vee y)(v \vee \overline{w})((v \wedge w) \vee \overline{y})(\overline{v} \vee z)(\overline{z} \vee \overline{v} \vee \overline{w}) \\ =\ &(v)(w)(v \vee \overline{w})(\overline{v} \vee z)(\overline{z} \vee \overline{v} \vee \overline{w}) \\ =\ &(v)(w)(z)(\overline{z} \vee \overline{v} \vee \overline{w}) = \mathbf{F} \ . \end{aligned}$$

Theorem 4. *Skolem and Herbrand functions generated by the clausal abstraction algorithm are correct.*

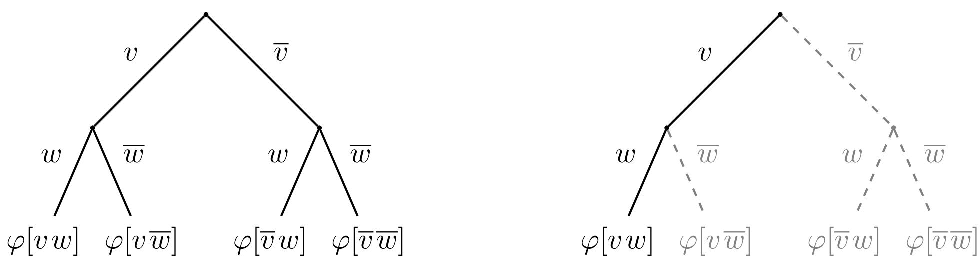
Proof. Let Φ be a true QBF over existential and universals variables $V_\exists$ and $V_\forall$, respectively, and let f be the Skolem function as described above. It holds that $f = \bigsqcup_{v \in V_\exists} f_v$ is well-formed by construction. Assume that f is not functionally correct. Thus, there is an assignment $\alpha_\forall$ of the universal variables $V_\forall$ such that $\alpha_\forall \vDash \neg\varphi[f]$. We show that f and $\alpha_\forall$ together lead to a root-to-leaf path in recursion tree such that all clauses in the matrix are satisfied. In detail, we build this path by a traversal of the recursion tree where at every node we take the leftmost choice such that

- at an existential node $\langle \exists X, \alpha_S \rangle$, we take the unique edge labeled with Sat and
- at an universal node $\langle \forall X, \alpha_S \rangle$, we take the leftmost edge labeled with $\mathsf{Sat}(\beta_S)$ such that the set of clauses in $\Phi|^{\forall X}_{\beta_S[\bot \mapsto \mathbf{F}]}$ is a superset of the clauses in $(\Phi|^{\forall X}_{\alpha_S})[\alpha_\forall|_X]$. Intuitively, the assignment $\alpha_\forall|_X$ satisfies more clauses than needed to show that the remaining subformula is true. This partial assignment β_S would have excluded $\alpha_\forall|_X$ under the assumption α_S in the refinement step of $\textsc{solve}_\forall(.)$ Note, that such an edge has to exist and all outgoing edges are labeled with Sat as otherwise, the universal node would not return Sat itself.

By construction, such a path exists and it is consistent with the Skolem function f due to Equation 10. Thus, f produces an assignment corresponding to $\alpha_\forall$ that satisfies the matrix, contradicting $\alpha_\forall \vDash \neg\varphi[f]$. Analogously for false QBFs. □

6. Integrating Partial Expansion

In this section, we continue our quest started in Section 4.3 for improved refinements for existential quantifiers. Expansion-based solving methods are based on the idea that a universal quantifier $\forall x. \varphi$ can be rewritten as the conjunction $\varphi[x \mapsto \mathbf{F}] \wedge \varphi'[x \mapsto \mathbf{T}]$ where



(a) A full expansion tree

(b) A partial expansion tree

Figure 2: A representation of full and partial expansion trees for formula $\forall v, w.\, \exists x, y.\, \varphi$, where $\varphi = (v \rightarrow x) \wedge (w \rightarrow y) \wedge (\overline{x} \vee \overline{y})$. The root-to-leaf paths represent a universal assignment $\alpha_{\{v,w\}}$ and the corresponding leaf node contains the propositional formula $\varphi[\alpha_{\{v,w\}}]$ expanded with $\alpha_{\{v,w\}}$. Both trees witness the unsatisfiability of $\forall v, w.\, \exists x, y.\, \varphi$.

x is eliminated by replacing it with $\mathbf{F}$ and $\mathbf{T}$ in the left and right conjunct, respectively, and by creating a copy of every variable in the right conjunct. By repeated application, a QBF can be transformed to a propositional formula. This type of *complete* expansion is for example implemented by the solvers Qubos [1], Quantor [11], and AIGSolve [68]. Consider, for example, the false QBF $\forall v, w.\, \exists x, y.\, (v \rightarrow x) \wedge (w \rightarrow y) \wedge (\overline{x} \vee \overline{y})$. Expanding v and w results into the unsatisfiable propositional formula $(x^{v\,w})(x^{v\,\overline{w}})(y^{v\,w})(y^{\overline{v}\,w})(\overline{x}^{v\,w} \vee \overline{y}^{v\,w})(\overline{x}^{\overline{v}\,w} \vee \overline{y}^{\overline{v}\,w})(\overline{x}^{v\,\overline{w}} \vee \overline{y}^{v\,\overline{w}})(\overline{x}^{\overline{v}\,\overline{w}} \vee \overline{y}^{\overline{v}\,\overline{w}})$. Here, we annotated variables a with the assignment α of the universal variables, written a^{α}. In Figure 2a we give a visual representation of the *full expansion tree*, that is, a tree whose root-to-leaf nodes represent all assignments α to universal variables.

Having to expand each and every universal variable and the resulting blow-up can be, however, avoided in many cases by a method called *partial expansion*. The idea is that already a subset of universal assignments can rule out the existence of any Skolem function. Instantiating the universal assignment $\{v \mapsto \mathbf{T}, w \mapsto \mathbf{T}\}$ in our example above leads an unsatisfiable formula $(x^{v\,w})(y^{v\,w})(\overline{x}^{v\,w} \vee \overline{y}^{v\,w})$. Thus, there can be no Skolem function for x and y if there is no assignment satisfying the matrix on a single universal assignment. In Figure 2b we give a visual representation of the *partial expansion tree*, that is, an expansion tree that does not necessarily contain all assignments. The solvers RAReQS [44] and Ijtihad [14] base their reasoning on partial expansion.

We are now going to show how to integrate partial expansion into the clausal abstraction algorithm. This integration combines the results of the correctness proof given in Section 4.2 and the function extraction presented in the previous section. The key insight is, that if $\textsc{solve}_\exists$ in Algorithm 3 determines that a quantified subformula $\Phi[\alpha'_S]$ is unsatisfiable, the witnessing Herbrand function corresponds to a *partial expansion tree* that can be used to strengthen the abstraction θ_X.

Notation. We start by providing necessary preliminaries and make the intuitive description given above more precise. For more details, we refer the reader to [45]. A *partial expansion tree* for QBF Φ with u universal quantifier blocks and matrix φ is a rooted tree $\mathcal{T}$ such that every path $p_0 \xrightarrow{\alpha_1} p_1 \cdots \xrightarrow{\alpha_u} p_u$ in $\mathcal{T}$ from the root p_0 to some leaf p_u has exactly u edges and each edge $p_{i-1} \xrightarrow{\alpha_i} p_i$ is labeled with an assignment α_i to the universal variables at universal level i. Each path in $\mathcal{T}$ is uniquely defined by its labeling. Let $\mathcal{T}$ be a partial expansion tree and $P = p_0 \xrightarrow{\alpha_1} p_1 \cdots \xrightarrow{\alpha_u} p_u$ be a path from the root p_0 to some leaf p_u. For an existential variable x we define $\mathit{expand\text{-}var}(P, x) = x^\alpha$ where x^α is a fresh variable and $\alpha = \left(\bigsqcup_{1 \leq i \leq u} \alpha_i\right)|_{dep(x)}$ is the universal assignment of the dependencies of x. For a propositional formula φ define $\mathit{expand}(P, \varphi)$ as instantiating φ with $\alpha_1, \ldots, \alpha_u$ and replacing every existential variable x by $\mathit{expand\text{-}var}(P, x)$. We define $\mathit{expand}(\mathcal{T}, \Phi)$ as the conjunction of all $\mathit{expand}(P, \varphi)$ for each root-to-leaf path P in $\mathcal{T}$.

Expansion Refinement. When the candidate verification algorithm returns $\mathsf{Unsat}(\beta_S)$ in line 8 in Algorithm 3, we extract the partial expansion tree $\mathcal{T}$ that witnesses the unsatisfiability result. Extracting partial expansion trees during solving is closely related to function extraction. Given an existential node $\langle \exists X, \alpha_S \rangle$ in the recursion tree (see Section 5), we build the partial expansion tree by traversing the subtree of $\langle \exists X, \alpha_S \rangle$ and record every universal assignment α at an edge labeled with Unsat. In the recursion tree depicted in Figure 1 and root node $\langle \exists \{v, w\}, \{s_i \mapsto 0 \mid 1 \leq i \leq 5\} \rangle$, the extracted partial expansion tree $\mathcal{T}$ contains the paths $p_0 \xrightarrow{\{x \mapsto \mathbf{F}\}} p_1$ and $p_0 \xrightarrow{\{x \mapsto \mathbf{T}\}} p_1'$ from root p_0 to the leaves p_1 and p_1'.

Finally, given the partial expansion tree $\mathcal{T}$, we build the clausal abstraction for every clause in the expansion formula $\mathit{expand}(\mathcal{T}, \Phi)$. The resulting clauses are added to the abstraction θ_X. Formally, after the clausal abstraction refinement in line 8, we update the abstraction by

$$\theta_X \leftarrow \theta_X \wedge \bigwedge_{C \in \mathit{expand}(\mathcal{T}, \Phi)} \mathit{clabs}_{\exists X}(C) \ .$$

Correctness of this refinement follows from the soundness of the partial expansion, i.e., replacing the matrix φ of some QBF Φ by $\varphi \wedge \mathit{expand}(\mathcal{T}, \Phi)$ preserves satisfiability for every expansion tree $\mathcal{T}$, and the correctness of the clausal abstraction. In the implementation, we can re-use the existent satisfaction variables s_i of some clause C_i for every corresponding expanded clause C_i^α as the literals bound by outer quantifier are equal, that is, $C_i^< = (C_i^\alpha)^<$.

7. Circuit Abstraction

A fundamental property of the PCNF game is that it is not dual for the two players: the existential player has to satisfy *all* clauses while the universal player tries to falsify some clause. This is especially visible in the underlying proof system: the refutation proof system is exponentially more succinct than the satisfaction proof system [47]. We propose a generalization of the clausal abstraction algorithm to propositional formulas in negation normal form (NNF), making the game effectively dual.

For this section, we assume an arbitrary (closed, prenex) QBF $\Phi = \mathcal{Q}X_1 \cdots \mathcal{Q}X_n. \varphi$ with quantifier prefix $\mathcal{Q}X_1 \cdots \mathcal{Q}X_n$ and propositional body φ in NNF.

7.1 Algorithm

Overview. The algorithm for solving QBF in negation normal form is in large parts a staightforward extension of the existential CNF algorithm shown in Section 4. The algorithm SOLVE, depicted in Algorithm 5, initializes the abstractions and returns the result of SOLVE-NNF, shown in Algorithm 6. SOLVE-NNF determines candidate assignments to

Algorithm 5 Abstraction Algorithm for QBF in negation normal form.

```
procedure SOLVE(Φ = 𝒬X. Ψ)
    initialize abstraction θ_Y and dual abstraction θ̄_Y for every quantifier 𝒬Y in Φ
    return SOLVE-NNF(𝒬X, Ψ, {s_i ↦ F | s_i ∈ S_X})
end procedure
```

the variables bound at that quantifier, which is then verified recursively, or gives a reason why there is no such assignment. In the negative case, this reason is excluded at an outer quantifier.

Going from CNF to NNF makes the algorithm *more uniform* and—at the same time—*more complex*, where the uniformity comes from the quantifiers' duality and the complexity arises from the less restrictive normal form. Taking both into account leads us to the most significant algorithmic contribution, the use of a *dual* abstraction $\overline{\theta}_X$ in conjunction with the abstraction θ_X seen in previous algorithms. The dual abstraction, whose name indicates that it is the abstraction for negation of the current quantifier, elegantly solves two issues that already arose in the previous algorithms but were much easier to handle for CNF. First, consider again the optimization discussed in Section 4.3 that improves the returned witness in the propositional case. In CNF, this was done by setting satisfaction variables to false whenever the current assignment (of existential variables) satisfies a clause. In NNF, we use the dual abstraction to generate those partial assignments from complete assignments of the satisfaction variables using a technique inspired by dual propagation [32, 35, 60]. Second, in the CNF algorithm Algorithm 3, we needed to project the partial assignment returned from the inner quantifier in case of a successful verification (line 8) as some of the clauses may be satisfied by the current assignment (of existential variables). In NNF, this is not merely a projection, but a transformation from one set of satisfaction variables to a (possibly) different set of satisfaction variables which can be efficiently implemented by the dual abstraction. Before going into details of algorithm SOLVE-NNF, we introduce the abstractions first.

Example 6. Consider again the QBF from Example 2 where the propositional formula φ is in negation normal form:

$$\exists x.\,\forall v,w.\,\exists y.\,\underbrace{(x \lor v \lor \overbrace{(y \land w)}^{\psi_3})}_{\psi_2} \land \underbrace{(\overline{x} \lor \overbrace{(v \land \overline{w})}^{\psi_5} \lor y)}_{\psi_4} \land \underbrace{(\overline{v} \lor w \lor \overline{y})}_{\psi_6} \tag{11}$$

Throughout this section, we use the naming of the subformulas as indicated in above and name $\psi_1 = \varphi$. Note, that the formula is true as witnessed by the Skolem functions $x = \mathbf{T}$ and $y(v, w) = \overline{v} \lor w$.

Abstraction *θ*. The abstraction θ_X is a propositional formula that represents, for every quantifier block $\mathcal{Q}X$, an over-approximation of the winning assignments α_X as well as the effect of the assignment α_X on the valuation of subformulas. The algorithm guarantees that whenever a candidate assignment α_X is generated using θ_X, all variables bound at outer quantifiers have a fixed assignment, and, thus, the propositional formula φ is partially evaluated.

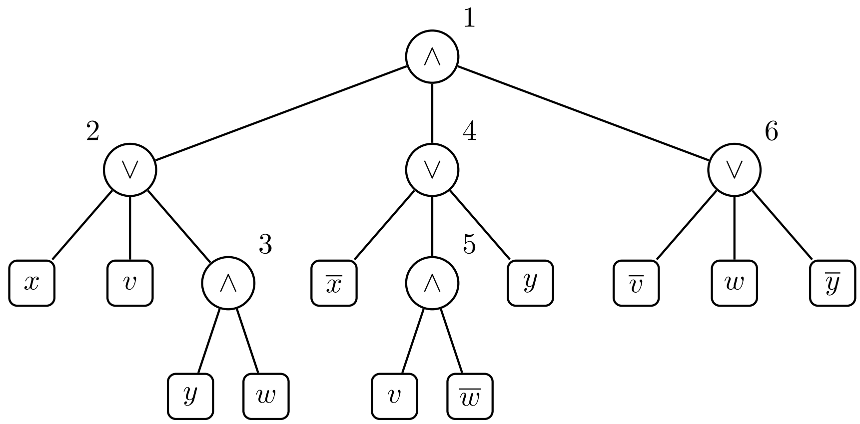


Figure 3: Visualization of the graph representation $\mathcal{G}_\varphi$ representation of $\varphi = (x \vee v \vee (y \wedge w)) \wedge (\overline{x} \vee (v \wedge \overline{w}) \vee y) \wedge (\overline{v} \vee w \vee \overline{y})$. The numbers on the non-terminal nodes represent the index i for the corresponding subformula ψ_i as shown in Example 6. To improve readability, some terminal nodes like y and w are drawn multiple times.

To facilitate working with arbitrary Boolean formulas, we start with introducing additional notation. Let $\mathcal{B}$ be the set of Boolean formulas and let $\mathit{sf}(\psi) \subset \mathcal{B}$ and $\mathit{dsf}(\psi) \subset \mathcal{B}$ be the set of all subformulas of ψ and the set of direct (or immediate) subformulas of ψ, respectively. Note that $\psi \in \mathit{sf}(\psi)$ but $\psi \notin \mathit{dsf}(\psi)$. For a propositional formula ψ, $\mathit{type}(\psi) \in \{\mathit{lit}, \vee, \wedge\}$ returns the Boolean connector if ψ is not a literal. For example, given $\psi = (x \vee v \vee (y \wedge w))$, the set of all subformulas is $\mathit{sf}(\psi) = \{(x \vee v \vee (y \wedge w)), x, v, (y \wedge w), y, w\}$, the set of direct subformulas is $\mathit{dsf}(\psi) = \{x, v, (y \wedge w)\}$, and the Boolean connector is $\mathit{type}(\psi) = \vee$. For every subformula ψ, we denote by $\overline{\psi}$ the dual subformula, that is, the formula where every quantifier, Boolean connector, and literal is negated. It holds that $\overline{\psi}$ is in NNF and that $\neg\overline{\psi}$ is equivalent to ψ.

We will explain the abstraction for quantifier $\exists X$ as a transformation of the graph representation of propositional formulas. A propositional formula ψ can be represented as a graph, where the nodes represent the Boolean connectives and the edges connect a formula with its direct subformulas. The leaves, i.e., terminal nodes, are the literals contained in ψ. Formally, the graph $\mathcal{G}_\varphi$ corresponding to some propositional formula φ is a pair $\langle V, E\rangle$, where $V = \mathit{sf}(\varphi)$ is the set of vertices and $E = V \times V$ is the edge relation such that $(\psi_i, \psi_j) \in E$ if, and only if, $\psi_j \in \mathit{dsf}(\psi_i)$. Figure 3 depicts the graph corresponding to the propositional part of the QBF presented in Example 6.

We define ψ° for $\circ \in \{<, \leq, =, \geq, >\}$ as the projection of ψ onto variables bound by outer ($<$), current ($=$), or inner ($>$) quantifiers with respect to $\mathcal{Q}X$, respectively. If the projected formula does not contain a literal, we return undefined $\perp$. Formally, we define ψ°

recursively (where we only recurse if the projection of a subformula is defined) as follows

$$\psi^\circ := \begin{cases} \bigwedge_{\substack{\psi_i \in dsf(\psi) \\ \psi_i^\circ \neq \bot}} \psi_i^\circ & \text{if } type(\psi) = \wedge \\ \bigvee_{\substack{\psi_i \in dsf(\psi) \\ \psi_i^\circ \neq \bot}} \psi_i^\circ & \text{if } type(\psi) = \vee \\ \psi & \text{if } \psi \text{ is a literal } l \text{ and } var(l) \text{ is bound at a quantifier level satisfying } \circ \\ \bot & \text{otherwise} \end{cases}$$

Applying this definition on our running example in Equation 11, we get, for example, $\psi_2^= = x$, $\psi_4^= = \overline{x}$ for quantifier $\exists x$; $\psi_2^\leq = x \vee v \vee w$, $\psi_4^\leq = \overline{x} \vee (v \wedge \overline{w})$ for quantifier $\forall v, w$; and $\psi_1^\leq = \varphi$ for quantifier $\exists y$.

We use the same kind of variables as in clausal abstraction to establish the interaction between abstractions: the variables X bound by the current quantifier and, additionally, the assumption and satisfaction variables A and S, respectively. The satisfaction variable s_i for some subformula $\psi_i \in sf(\varphi)$ represents the effect of variables V bound at outer quantifier on ψ_i. To quantify this effect, we have to distinguish whether ψ_i is a disjunctive ($type(\psi_i) = \vee$) or conjunctive ($type(\psi_i) = \wedge$) formula. In the disjunctive case, assigning s_i to true implies that ψ_i evaluates to true given the outer variable assignment α_V. This is a straightforward generalization of the existential abstraction for clauses (see Section 4). In case ψ_i is conjunctive, a positive assignment of s_i means that the conjunct is not yet falsified, that is, ψ_i does not evaluate to false given the outer variable assignment α_V. We combine both cases by saying that ψ_i is assigned *positively* with respect to the current quantifier. Since the valuation of the variables X bound by the current quantifier has an influence on the valuation of subformulas as well, we use an assumption variable a_i to represent the effect of the combined assignments α_X and α_S. The intended semantics is that a_i is set to false only if ψ_i is assigned positively at this quantifier (by assignment $\alpha_X \dot\sqcup \alpha_V$).

Before formally defining the abstraction, we discuss the underlying derivation steps on Example 6.

Example 7. The abstraction θ_X quantifies the effect of valuations of variables X on the satisfaction of subformulas. We derive the abstraction by transforming the graph representation of φ and $\overline{\varphi}$ for existential and universal quantifiers, respectively. This transformation is visualized in Figure 4. As a first step, we remove all subformulas ψ which are only influenced by inner quantifiers, i.e., every ψ that is not contained in $\varphi^\leq$. For example, ψ_6 does not contain x, thus, the whole subformula is removed from φ for quantifier $\exists x$.

Then, we replace all maximal subformulas with the property $\psi^< = \psi$ by satisfaction variables s_i in a top-down way. Consider the innermost quantifier $\exists y$ and subformula ψ_4 with $dsf(\psi_4) = \{\overline{x}, \psi_5, y\}$. For the former two, x and ψ_5, it holds that $x^< = x$ and $\psi_5^\leq = \psi_5$, thus, both are replaced with the satisfaction variable s_4.

In the innermost quantifier $\exists y$, this already adequately describes the abstraction, for every other quantifier we have to define the assumption variables. For example at quantifier $\exists x$, an assignment to x can either satisfy ψ_2 or ψ_4, but not both, thus, the other formula is assumed to be satisfied by an inner quantifier. We define an assumption variable for

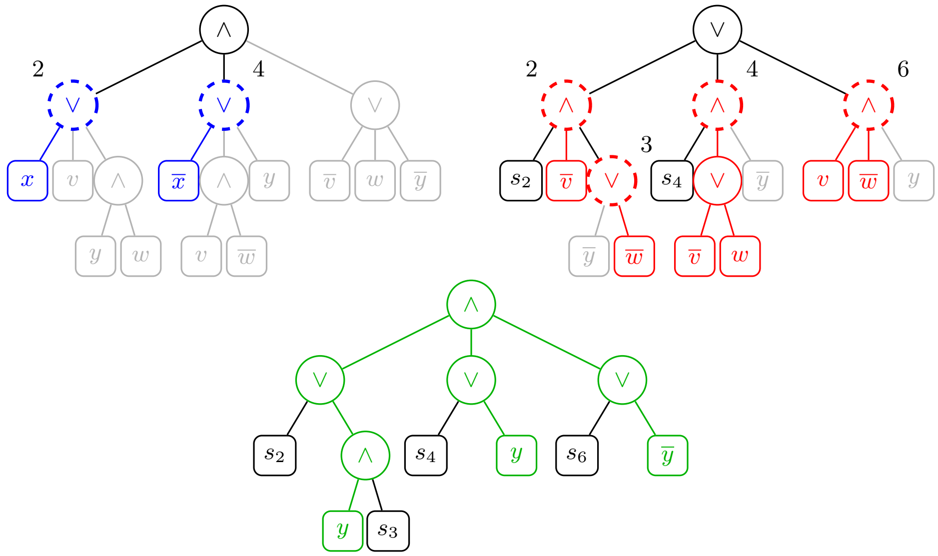


Figure 4: Abstraction for quantifiers $\exists x$, $\forall v, w$, and $\exists y$. The grayed out subformulas are only influenced by inner variables. The colored parts indicate continuous subformulas with $\psi = \psi^{\leq}$. The dashed subformulas indicate placement of assumption variables.

every subformula $\psi_i \in sf(\varphi)$ such that there exist direct subformulas ψ_j and ψ_k such that $\psi_j = \psi_j^{\leq}$ and $\psi_k \neq \psi_k^{\leq}$. Intuitively, for these subformulas ψ_i, there is a direct influence by $\psi_j = \psi_j^{\leq}$ and the value of ψ_i is not guaranteed to be determined after the current quantifier as there is some influence by inner variables $\psi_k \neq \psi_k^{\leq}$. This can be seen at our example at quantifier $\forall v, w$: we need to add an assumption variable to ψ_3 as $\overline{y} \in dsf(\psi_3)$ but not to ψ_5 as $\psi_5^{=} = \psi_5$. Lastly, given some quantifier alternation $\mathcal{Q}X.\, \overline{\mathcal{Q}}Y$, there is a one-to-one correspondence between the assumption variables A_X of quantifier $\mathcal{Q}X$ and the satisfaction variables S_Y of quantifier Y.

Using the intuition of the interface variables and the determinacy of subformulas, we are now going to define the abstraction formally. In this definition, we take advantage of the duality by only defining the abstraction for existential quantifiers. The abstraction for universal quantifiers is then the abstraction for the negated formula $\overline{\Phi}$. Let us fix some existential quantifier $\exists X$ and some subformula ψ_i of φ. The abstraction θ_X is defined as

$$\theta_X = \begin{cases} enc(\varphi) & \exists X \text{ is the innermost quantifier} \\ \displaystyle\bigwedge_{\substack{\psi_i \in sf(\varphi) \\ \exists \psi_j, \psi_k \in dsf(\psi_i) \text{ with } \psi_j = \psi_j^{\leq} \wedge \psi_k \neq \psi_k^{\leq}}} a_i \vee enc(\psi_i) & \text{otherwise} \end{cases} \tag{12}$$

For the innermost quantifier $\exists X.\, \varphi$, we encode φ using *enc* defined below. In all other cases, we define the implication that setting an assumption variable a_i to false is only possible if the formula $enc(\psi_i)$ is satisfied. $enc(\psi_i)$ considers only subformulas of ψ_i which do not contain inner variables and where outer variables are replaced by their respective satisfaction literals. Formally, the abstraction for ψ_i is defined as

$$enc(\psi_i) = \begin{cases} \bigwedge\limits_{\substack{\psi_j \in dsf(\psi_i) \\ \psi_j = \psi_j^{\leq}}} enc_{\psi_i}(\psi_j) & \text{if } type(\psi_i) = \wedge \\ \bigvee\limits_{\substack{\psi_j \in dsf(\psi_i) \\ \psi_j = \psi_j^{\leq}}} enc_{\psi_i}(\psi_j) & \text{if } type(\psi_i) = \vee \end{cases} \tag{13}$$

where the direct subformulas ψ_j of ψ_i are transformed as follows

$$enc_{\psi_i}(\psi_j) = \begin{cases} \psi_j & \text{if } \psi_j = \psi_j^{=} \\ s_i & \text{if } \psi_j = \psi_j^{<} \\ enc(\psi_j) & \text{otherwise, i.e., } \psi_j = \psi_j^{\leq} \end{cases} \tag{14}$$

We carefully dissect the definitions in order to map them to the intuitions mentioned above. The function $enc(\psi_i)$ builds the abstraction for subformula ψ_i depending on the Boolean connector $type(\psi_i) \in \{\wedge, \vee\}$. Further, *enc* considers only those direct subformulas ψ_j of ψ_i, which are solely influenced by the current or outer variables, i.e., $\psi_j = \psi_j^{\leq}$. The encoding of direct subformulas $enc_{\psi_i}(\psi_j)$ distinguishes three cases. If ψ_j contains only variables X, that is, $\psi_j = \psi_j^{=}$, then the result of $enc_{\psi_i}(\psi_j) = \psi_j$ is the formula ψ_j itself. If ψ_j contains only outer variables, that is, $\psi_j = \psi_j^{<}$, then the result of $enc_{\psi_i}(\psi_j) = s_i$ is the satisfaction variable s_i. Finally, if ψ_j contains both types of variables, we apply *enc* on ψ_j.

The abstraction for a universal quantifier $\forall X$ and the dual abstraction $\overline{\theta}_X$ of quantifier $\exists X$ are both defined as the abstraction for $\exists X$ with respect to propositional formula $\overline{\varphi}$. As discussed above, satisfaction and assumption variables are not exposed for every subformula $\psi_i \in sf(\varphi)$. For the given abstraction, we define the set of interface variables for quantifier $\mathcal{Q}X$ as

$$\begin{aligned} A_X &= \{a_i \mid \psi_i \in sf(\varphi) \wedge \exists \psi_j, \psi_k \in dsf(\psi_i).\, \psi_j = \psi_j^{\leq} \wedge \psi_k \neq \psi_k^{\leq}\} \text{ and} \\ S_X &= \{s_i \mid \psi_i \in sf(\varphi) \wedge \exists \psi_j, \psi_k \in dsf(\psi_i).\, \psi_j = \psi_j^{<} \wedge \psi_k \neq \psi_k^{<}\} \ . \end{aligned}$$

This means that for some quantifier alternation $\mathcal{Q}X.\, \overline{\mathcal{Q}}Y$ the sets A_X and S_Y represent the same subformulas, i.e., $a_i \in A_X$ if, and only if, $s_i \in S_Y$.

The algorithm makes progress by refining the abstraction during the execution of the algorithm. Such a refinement excludes wrong assumptions, i.e., assumptions corresponding to a losing assignment for the variables of the respective quantifier block. Given such a set of assumptions $L \subseteq A$, the refinement is represented by the clause

$$\bigvee_{a_i \in L} \overline{a_i} \ . \tag{15}$$

Algorithm. Algorithm 6 shows the recursive QBF solving algorithm SOLVE-NNF. It decides the problem whether the quantified subformula $\mathcal{Q}X.\,\Phi$ of Φ for $\mathcal{Q} \in \{\forall, \exists\}$ is satisfiable under the condition that the propositional formula φ is partially evaluated according to the assignment α_S that abstracts the outer variable assignment. Note that due to duality, the

Algorithm 6 Algorithm for solving quantified formulas in NNF.

1: **procedure** SOLVE-NNF($\mathcal{Q}X,\ \Phi,\ \alpha_{S_X}$)
2:     **loop**
3:         **match** $\langle \text{SAT}(\theta_X, \alpha_{S_X}),\ \ \Phi\rangle$ **as** ▷ assume outer variable assignment
4:             $\langle \mathsf{Sat}(\alpha),\ \ \overline{\mathcal{Q}}Y.\,\Psi\rangle \Rightarrow$
5:                 $\alpha_{S_Y} \leftarrow \{s_i \mapsto \alpha(a_i) \mid a_i \in A_X\}$ ▷ update subformula valuation
6:                 **match** SOLVE-NNF($\overline{\mathcal{Q}}Y,\ \Psi,\ \alpha_{S_Y}$) **as** ▷ recursive verification
7:                     $\mathsf{Sat}_{\mathcal{Q}}(\beta_{S_Y}) \Rightarrow\ \ \overline{\theta}_X \leftarrow \overline{\theta}_X \wedge \bigvee_{s_i \in \beta^0_{S_Y}} \overline{a}_i$ ▷ refine $\overline{\theta}_X$
8:                         **return** $\mathsf{Sat}_{\mathcal{Q}}(\text{OPTIMIZE}(\alpha|_X, \alpha_{S_X}))$
9:                     $\mathsf{Unsat}_{\mathcal{Q}}(\beta_{S_Y}) \Rightarrow\ \ \theta_X \leftarrow \theta_X \wedge \bigvee_{s_i \in \beta^1_{S_Y}} \overline{a}_i$ ▷ refine θ_X
10:             $\langle \mathsf{Sat}(\alpha),\ \ \_\rangle \Rightarrow$ **return** $\mathsf{Sat}_{\mathcal{Q}}(\text{OPTIMIZE}(\alpha|_X, \alpha_{S_X}))$ ▷ propositional
11:             $\langle \mathsf{Unsat}(\beta_{S_X}),\ \ \_\rangle \Rightarrow$ **return** $\mathsf{Unsat}_{\mathcal{Q}}(\beta_{S_X})$
12:     **end loop**
13: **end procedure**
14: **procedure** OPTIMIZE($\alpha_X,\ \alpha_{S_X}$)
15:     **match** SAT($\overline{\theta}_X, \alpha_X \sqcup \overline{\alpha}_{S_X}$) **as**
16:         $\mathsf{Unsat}(\beta) \Rightarrow$ **return** $\overline{\beta}|_{S_X}$ ▷ $\overline{\beta}|_{S_X} \sqsubseteq \alpha_{S_X}$
17: **end procedure**

satisfiability and unsatisfiability are interpreted with respect to the current quantifier, that is, we define

$$\mathsf{Sat}_{\mathcal{Q}} = \begin{cases} \mathsf{Sat} & \text{if } \mathcal{Q} = \exists \\ \mathsf{Unsat} & \text{if } \mathcal{Q} = \forall \end{cases} \qquad \text{and} \qquad \mathsf{Unsat}_{\mathcal{Q}} = \begin{cases} \mathsf{Unsat} & \text{if } \mathcal{Q} = \exists \\ \mathsf{Sat} & \text{if } \mathcal{Q} = \forall \end{cases}.$$

For sake of simplicity, we base our explanation on existential quantifier in the following. The algorithm repeatedly generates candidate assignments by means of the abstraction θ_X (line 3). If the abstraction returns Unsat, there is no satisfiable assignment with respect to the assignment α_S of satisfaction variables, thus, the algorithm returns $\mathsf{Unsat}_{\mathcal{Q}}$ as well (line 11). Further, the reason for the unsatisfiability result is given, represented by the returned partial assignment β_{S_X}. If the abstraction returns Sat with assignments α_A and α_X, we distinguish two cases. The first case is the base case of the recursion, that is, the inner formula is quantifier-free (line 10). The algorithm returns $\mathsf{Sat}_{\mathcal{Q}}$ and the partial assignment, generated by the algorithm OPTIMIZE, indicating which subformulas have to be positively assigned by outer quantifier such that the assignment α_X satisfies φ. Lastly, assume that the inner subformula is quantified. In this case, we compute the subformulas of φ that the combination of α_X and α_S assign positively (line 5) and continue with the recursive verification. In the positive case, the partial assignment β_{S_Y} (line 7) indicates the required positively assigned subformulas. As this witnesses the unsatisfiability of the negated formula, the dual abstraction $\overline{\theta}_X$ is refined with β_{S_Y} before translating the assignment β_{S_Y} to an assignment β_{S_X} using OPTIMIZE in line 8. In case it is negative, the abstraction θ_X is refined by enforcing that some negatively assigned subformulas is assigned postively, before continuing with the next iteration.

The algorithm optimize implements dual propagation. The dual abstraction $\overline{\theta}_X$ is a representation of the possible assignments of the negated quantifier $\overline{\mathcal{Q}}X$. Thus, assuming the positively verified assignments α_X and α_S (lines 8 and 10) lead to unsatisfiability of $\overline{\theta}_X$. Note, that we have to negate α_S due to the way the abstraction is built (a formal justification is given in Section 7.2). The return value β, that is, the failed assumptions projected onto variables S_X, represents a set of subformulas $\{\psi_i \mid s_i \in \beta^1_{S_X}\}$ that needs to be assigned positively such that the quantifier $\mathcal{Q}X$ has a satisfiable assignment.

Example 8. Consider again the formula given in Example 2:

$$\exists x.\, \forall v, w.\, \exists y.\, \underbrace{(x \vee v \vee \overbrace{(y \wedge w)}^{\psi_3})}_{\psi_2} \wedge \underbrace{(\overline{x} \vee \overbrace{(v \wedge \overline{w})}^{\psi_5} \vee y)}_{\psi_4} \wedge \underbrace{(\overline{v} \vee w \vee \overline{y})}_{\psi_6}$$

We give the abstractions as discussed in Example 7 in propositional form as

$$\begin{aligned}
\theta_{\{x\}} &= (a_2 \vee x)(a_4 \vee \overline{x}),\\
\theta_{\{v,w\}} &= (a_2 \vee (s_2 \wedge \overline{v}))(a_3 \vee \overline{w})(a_4 \vee (s_4 \wedge (\overline{v} \vee w)))(a_6 \vee (v \wedge \overline{w})),\\
\overline{\theta}_{\{v,w\}} &= (a_2 \vee s_2 \vee v)(a_3 \vee w)(a_4 \vee s_4 \vee (v \wedge \overline{w}))(a_6 \vee \overline{v} \vee w),\\
\theta_{\{y\}} &= (s_2 \vee (y \wedge s_3))(s_4 \vee y)(s_6 \vee \overline{y}), \text{ and}\\
\overline{\theta}_{\{y\}} &= (s_2 \wedge (\overline{y} \vee s_3)) \vee (s_4 \wedge \overline{y}) \vee (s_6 \wedge y).
\end{aligned}$$

We give a possible execution of algorithm solve. To improve readability, we use the propositional representation for assignments.

- solve-nnf$(\exists x,\, \forall v, w.\, \exists y.\, \varphi,\, \{\})$
- sat$(\theta_{\{x\}}) = \mathsf{Sat}(\overline{x}\, a_2 \overline{a_4})$
- $\alpha_{S_{\{v,w\}}} = s_2 \overline{s_4}$
  - solve-nnf$(\forall v, w,\, \exists y.\, \varphi,\, \alpha_{S_{\{v,w\}}})$
  - sat$(\theta_{\{v,w\}},\, \alpha_{S_{\{v,w\}}}) = \mathsf{Sat}(\overline{v}\, \overline{w}\, \overline{a_2}\, \overline{a_3} a_4 a_6)$
  - $\alpha_{S_{\{y\}}} = \overline{s_2}\, \overline{s_3} s_4 s_6$
    - solve-nnf$(\exists y,\, \varphi,\, \alpha_{S_{\{y\}}})$
    - solve$(\theta_{\{y\}},\, \alpha_{S_{\{y\}}}) = \mathsf{Unsat}(\overline{s_2}\, \overline{s_3})$
    - **return** $\mathsf{Unsat}(\overline{s_2}\, \overline{s_3})$
  - $\overline{\theta}'_{\{v,w\}} = \overline{\theta}_{\{v,w\}} \wedge (\overline{a_2} \vee \overline{a_3}) = (s_2 \vee v \vee w)\ [\ldots]$
  - solve$(\overline{\theta}'_{\{vw\}},\, \overline{v}\, \overline{w}\, \overline{\alpha}_{S_{\{v,w\}}}) = \mathsf{Unsat}(\overline{v}\, \overline{w}\, \overline{s_2})$
  - **return** $\mathsf{Unsat}(s_2)$
- $\theta'_{\{x\}} = \theta_{\{x\}} \wedge \overline{a_2}$
- sat$(\theta'_{\{x\}}) = \mathsf{Sat}(x\, \overline{a_2} a_4)$

- $\alpha'_{S_{\{v,w\}}} = \overline{s_2} s_4$
  - SOLVE-NNF($\forall v, w,\, \exists y.\, \varphi,\, \alpha'_{S_{\{v,w\}}}$)
  - SAT($\theta_{\{v,w\}},\, \alpha'_{S_{\{v,w\}}}$) = $\mathsf{Sat}(\overline{v}\,\overline{w}\,a_2\overline{a_3}\,\overline{a_4}a_6)$
  - $\alpha'_{S_{\{y\}}} = s_2\overline{s_3}\,\overline{s_4}s_6$
    - SOLVE-NNF($\exists y,\, \varphi,\, \alpha'_{S_{\{y\}}}$)
    - SOLVE($\theta_{\{y\}},\, \alpha'_{S_{\{y\}}}$) = $\mathsf{Sat}(y)$
    - SOLVE($\overline{\theta}_{\{y\}},\, y\,\overline{\alpha}'_{S_{\{y\}}}$) = $\mathsf{Unsat}(y\,\overline{s_2}\,\overline{s_6})$
    - **return** $\mathsf{Sat}(s_2 s_6)$
  - $\theta'_{\{v,w\}} = \theta_{\{v,w\}} \land (\overline{a_2} \lor \overline{a_6})$
  - SAT($\theta_{\{v,w\}},\, \alpha'_{S_{\{v,w\}}}$) = $\mathsf{Sat}(v\,\overline{w}\,a_2\overline{a_3}a_4\overline{a_6})$
  - $\alpha''_{S_{\{y\}}} = s_2\overline{s_3}s_4\overline{s_6}$
    - SOLVE-NNF($\exists y,\, \varphi,\, \alpha''_{S_{\{y\}}}$)
    - SOLVE($\theta_{\{y\}},\, \alpha''_{S_{\{y\}}}$) = $\mathsf{Sat}(\overline{y})$
    - SOLVE($\overline{\theta}_{\{y\}},\, \overline{y}\,\overline{\alpha}''_{S_{\{y\}}}$) = $\mathsf{Unsat}(\overline{y}\,\overline{s_2}\,\overline{s_4})$
    - **return** $\mathsf{Sat}(s_2 s_4)$
  - $\theta''_{\{v,w\}} = \theta'_{\{v,w\}} \land (\overline{a_2} \lor \overline{a_4})$
  - SAT($\theta''_{\{v,w\}},\, \alpha'_{S_{\{v,w\}}}$) = $\mathsf{Unsat}(\overline{s_2})$
  - **return** $\mathsf{Sat}(\overline{s_2})$
- SOLVE-NNF($\exists x,\, \forall v, w.\, \exists y.\, \varphi,\, \{\}$) returns Sat

7.2 Correctness

The proof of correctness requires the same high level argumentation as the correctness proof for the prenex conjunctive normal form algorithm in Section 4.2. The argumentation over the abstraction and negation normal form formulas is, however, much more sophisticated than the argumentation over clauses in a matrix. Thus, in this section, we give a rigorous argumentation for soundness and completeness, even though there is some repetition and overlap with Section 4.2. Remember, that we fixed a QBF $\Phi = \mathcal{Q}X_1 \cdots \mathcal{Q}X_n.\, \varphi$ with quantifier prefix $\mathcal{Q}X_1 \cdots \mathcal{Q}X_n$ and propositional body φ in NNF. Further, we assume that $\psi_1, \ldots, \psi_m$ are the non-literal subformulas of φ.

Before going into detail, we outline the structure of this section. First, we establish a relation between assignments of satisfaction variables α_S and their effect on the QBF, analogously to Section 4.2. For some quantifier alternation $\mathcal{Q}X.\, \overline{\mathcal{Q}}Y$, we show how assignments α_{S_X} with respect to quantifier $\mathcal{Q}X$ are related to assignments α_{S_Y} w.r.t. quantifier $\overline{\mathcal{Q}}Y$. Afterwards, we establish statements over the abstractions, the first (Lemma 11) covering the base case of the structural induction. Furthermore, we show that the abstractions θ_X and $\overline{\theta}_X$ are effectively dual, which leads to the correctness of the dual propagation in Lemma 13. The actual proof of correctness is carried out in Lemma 14.

Duality in NNF representation. To match the assignment of the satisfaction variables α_S with the corresponding valuation of the propositional formula φ, we define a partial function that maps subformulas of φ to a Boolean valuation $\mathbb{B}$ or undefined $\bot$. We use the convention to write such subformula valuation functions as $\beta_\varphi \colon sf(\varphi) \to \mathbb{B}_\bot$, i.e., we index the partial function by a propositional formula. Then, similar to the correctness proof of clausal abstraction in Section 4.2, we define an operation $\Phi|_{\beta_\varphi}^{\mathcal{Q}X}$, for a QBF Φ, quantifier $\mathcal{Q}X$, and subformula valuation β_φ, as the QBF with the same prefix as Φ with propositional formula φ' resulting from replacing subformulas ψ_i by their valuation $\beta_\varphi(\psi_i)$ if it is defined. Potentially occurring free variables, which were in the original QBF variables bound by outer quantifiers, are removed by this operation.

Formally, the propositional part of $\Phi|_{\beta_\varphi}^{\mathcal{Q}X}$ is defined as the partial evaluation of φ according to the subformula valuation β_φ. Therefore, we use a partial evaluation function $\mathit{parteval}(\psi, \beta_\varphi)$ that maps a propositional formula ψ and a subformula valuation β_φ to a propositional formula. It is defined as

$$\mathit{parteval}(\psi, \beta_\varphi) = \begin{cases} \beta_\varphi(\psi) & \text{if } \beta_\varphi(\psi) \neq \bot \\ \displaystyle\bigwedge_{\substack{\psi' \in dsf(\psi) \\ \mathit{parteval}(\psi', \beta_\varphi) \neq \bot}} \mathit{parteval}(\psi', \beta_\varphi) & \text{if } \mathit{type}(\psi) = \wedge \\ \displaystyle\bigvee_{\substack{\psi' \in dsf(\psi) \\ \mathit{parteval}(\psi', \beta_\varphi) \neq \bot}} \mathit{parteval}(\psi', \beta_\varphi) & \text{if } \mathit{type}(\psi) = \vee \\ \psi & \text{if } \mathit{type}(\psi) = \mathit{lit} \text{ and } \psi \text{ is bound} \\ \bot & \text{otherwise} \end{cases}$$

Lastly, we need to define the subformula valuation function corresponding to some assignment of satisfaction variables α_S. An assignment of satisfaction variables α_S represents the subformula valuation $\beta_\varphi \coloneqq \mathit{sfval}_\varphi(\alpha_S)$ where $\mathit{sfval}_\varphi(\alpha_S)$ is defined as

$$\mathit{sfval}_\varphi(\alpha_S)(\psi_i) = \begin{cases} \mathbf{T} & \text{if } s_i \in \mathrm{dom}(\alpha_S) \wedge (\mathit{type}(\psi_i) = \vee) \wedge \alpha_S(s_i) = \mathbf{T} \\ \mathbf{F} & \text{if } s_i \in \mathrm{dom}(\alpha_S) \wedge (\mathit{type}(\psi_i) = \wedge) \wedge \alpha_S(s_i) = \mathbf{F} \\ \bot & \text{otherwise} \end{cases} \tag{16}$$

Then, we define the shorthand notation $\Phi|_{\alpha_S}^{\mathcal{Q}X}$ as $\Phi|_{\beta_\varphi}^{\mathcal{Q}X}$, where $\beta_\varphi \coloneqq \mathit{sfval}_\varphi(\alpha_S)$.

Many times in this section, we will argue about *duality*. To make this reasoning precise, we begin with a formal justification using two lemmata. Recall that we denote by $\overline{\beta}$ the complement of the partial assignment β. The following lemma states that an assignment α_{S_X} for Φ corresponds to the negated assignment $\overline{\alpha}_{S_X}$ in the negated formula $\overline{\Phi}$.

Lemma 8 (Duality)**.** *Let Φ be a QBF with propositional formula φ, let $\mathcal{Q}X$ be some quantifier of Φ, and let α_{S_X} be an assignment to the satisfaction variables. $\Phi|_{\alpha_{S_X}}^{\mathcal{Q}X}$ is true if, and only if, $\overline{\Phi}|_{\overline{\alpha}_{S_X}}^{\overline{\mathcal{Q}}X}$ is false.*

Proof. Let $\beta_\varphi \coloneqq \mathit{sfval}_\varphi(\alpha_{S_X})$ and let $\beta_{\overline{\varphi}} \coloneqq \mathit{sfval}_{\overline{\varphi}}(\overline{\alpha}_{S_X})$. It holds that $\overline{\beta}_\varphi = \beta_{\overline{\varphi}}$ by the definition of *sfval* in Equation 16. For every QBF Φ it holds that Φ is true if, and only if, $\overline{\Phi}$ is false. Together, this shows that $\Phi|_{\alpha_{S_X}}^{\mathcal{Q}X}$ is true iff $\overline{\Phi}|_{\overline{\alpha}_{S_X}}^{\overline{\mathcal{Q}}X}$ is false. ☐

In the correctness proof below, we will argue over optimal assumption assignments for some quantifier $\mathcal{Q}X$, that is, assignments of assumption variables $\alpha^*_{A_X}$ that are minimal with respect to the number of assumptions $\alpha^*_{A_X}(a_i) = \mathbf{T}$. The following lemma establishes this form of reasoning for quantifier alternations by proving equisatisfiability between $(\Phi|^{\mathcal{Q}X}_{\alpha_{S_X}})[\alpha_X]$ and $\Phi|^{\overline{\mathcal{Q}}Y}_{\alpha^*_{S_Y}}$ for some "optimal" α_{S_Y} constructed from α_{S_X} and α_X analogously to Lemma 5. In the proof, we argue over *consistency* of *complete* subformula assignments, which means that the assignment respects the propositional formula. A complete subformula assignment $\alpha_\varphi \colon sf(\varphi) \to \mathbb{B}$ is *consistent*, if and only if, for every (non-literal) subformula ψ_i of φ it holds that

$$\alpha_\varphi(\psi_i) = \begin{cases} \bigwedge_{\psi_j \in dsf(\psi_i)} \alpha_\varphi(\psi_j) & \text{if } type(\psi_i) = \wedge \\ \bigvee_{\psi_j \in dsf(\psi_i)} \alpha_\varphi(\psi_j) & \text{otherwise} \end{cases}$$

Lemma 9. *Let $\mathcal{Q}X.\,\overline{\mathcal{Q}}Y$ be a quantifier alternation of a QBF Φ with propositional formula φ and let α_X and α_{S_X} be assignments. Further, let $\alpha^*_{S_Y}$ be defined such that $\alpha^*_{S_Y}(s_i) = \mathbf{F}$ if, and only if, $\alpha_X \dot\sqcup \alpha_{S_X} \vDash enc(\psi_i)$ (for quantifier $\mathcal{Q}X$). It holds that $(\Phi|^{\mathcal{Q}X}_{\alpha_{S_X}})[\alpha_X]$ and $\Phi|^{\overline{\mathcal{Q}}Y}_{\alpha^*_{S_Y}}$ are equisatisfiable.*

Proof. We prove the statement for quantifier alternations of the form $\exists X.\,\forall Y$, the case $\forall X.\,\exists Y$ then follows by Lemma 8. The quantified formulas $(\Phi|^{\exists X}_{\alpha_{S_X}})[\alpha_X]$ and $\Phi|^{\forall Y}_{\alpha^*_{S_Y}}$ have the same quantifier prefix (starting with $\forall Y$) and equisatisfiable propositional formula. We show the latter by proving equality over the corresponding subformula assignments. Let $\beta_\varphi \coloneqq sfval_\varphi(\alpha_{S_X})$. We augment β_φ with α_X, that is, we define $\beta'_\varphi \coloneqq \beta_\varphi \sqcup \alpha_X$. Let $\beta_{\overline{\varphi}} = sfval_{\overline{\varphi}}(\alpha^*_{S_Y})$ be the subformula valuation corresponding to $\alpha^*_{S_Y}$. Note that $sfval_{\overline{\varphi}}(\alpha^*_{S_Y})$ is defined with respect to negated subformulas, that is, $\overline{\psi_i} \in \overline{\varphi}$ as it corresponds to a universal quantifier $\forall Y$ (and is thus equivalent to the existential quantifier $\exists Y$ over the dual propositional formula $\overline{\varphi}$). We show that the assignments β'_φ and $\beta_{\overline{\varphi}}$ are dual with respect to the satisfaction variables for quantifier $\forall Y$. As the assignment α_X may propagate subformula valuations beyond the boundaries given by the satisfaction variables, we prove the following strengthening: For every complete and consistent extension α_φ of β'_φ ($\beta'_\varphi \sqsubseteq \alpha_\varphi$) and every $s_i \in S_Y$ it holds that $\alpha_\varphi(\psi_i) = \overline{\beta}_{\overline{\varphi}}(\overline{\psi_i})$ if $\beta_{\overline{\varphi}}(\overline{\psi_i}) \neq \bot$.

Let $\beta_{\overline{\varphi}}(\overline{\psi_i}) = \mathbf{T}$ (analogous for $\beta_{\overline{\varphi}}(\overline{\psi_i}) = \mathbf{F}$). Then, by definition of $sfval_{\overline{\varphi}}(\alpha^*_{S_Y})(\overline{\psi_i})$ in Equation 16 it holds that $type(\overline{\psi_i}) = \vee$ and $\alpha^*_{S_Y}(\overline{s_i}) = \mathbf{T}$. By the definition of $\alpha^*_{S_Y}$, we know that $\alpha_X \dot\sqcup \alpha_{S_X} \nvDash enc(\psi_i)$. By the definition of the abstraction θ_X, for every $s_i \in S_Y$, there is a $\psi_j \in dsf(\psi_i)$ such that $\psi_j = \psi_j^<$, that is, ψ_j is only influenced by outer variables (with respect to X). A recursive argument over $enc(\psi_i)$ shows that, $\alpha_X \dot\sqcup \alpha_{S_X} \nvDash enc(\psi_i)$ implies that for every complete and consistent subformula valuation $\alpha_\varphi(\psi_i) = \mathbf{F}$ has to hold.

Further, for every complete and consistent extension $\alpha_{\overline{\varphi}}$ of $\beta_{\overline{\varphi}}$ with the same variable assignments as α_φ ($\alpha_{\overline{\varphi}}(v) = \alpha_\varphi(v)$ for every bound variable v), it holds that $\alpha_{\overline{\varphi}}(\overline{\psi_i}) = \overline{\alpha}_\varphi(\psi_i)$ by duality and the previous statement. ☐

If the assignments are not optimal, there is a monotonicity property on the satisfaction assignments stated below.

Lemma 10 (Monotonicity of α_{S_X})**.** *Let $\mathcal{Q}X$ be a quantifier of QBF Φ and let α_{S_X} be an assignment such that $\Phi|^{\mathcal{Q}X}_{\alpha_{S_X}}$ is winning for $\mathcal{Q}X$. For every α'_{S_X} with $\alpha^+_{S_X} \sqsubseteq {\alpha'_{S_X}}^+$ it holds that $\Phi|^{\mathcal{Q}X}_{\alpha'_{S_X}}$ is winning for $\mathcal{Q}X$.*

Proof. We prove the statement for $\exists X$, the universal case is analogous. Let α_{S_X} be given such that $\Phi|^{\exists X}_{\alpha_{S_X}}$ is winning for $\exists X$. Further, choose some arbitrary α'_{S_X} with $\alpha^+_{S_X} \sqsubseteq {\alpha'_{S_X}}^+$. The subformula valuations β_φ and β'_φ corresponding to α_{S_X} and α'_{S_X}, respectively, are monotone as well: If $\beta_\varphi(\psi_i) = \mathbf{T}$ it follows that $\beta'_\varphi(\psi_i) = \mathbf{T}$ by definition in Equation 16. $\square$

Reasoning over abstractions $\boldsymbol{\theta_X}$ and $\boldsymbol{\overline{\theta}_X}$. In this part, we focus on the two types of abstractions used in the algorithm. First, we have a formal statement regarding equisatisfiability of the innermost abstraction and the circuit representation for a given assignment of the satisfaction variables α_S, similar to Lemma 4.1.

Lemma 11. *Let Φ be a QBF with propositional formula φ, let $\exists X$ be the innermost quantifier, and let α_{S_X} be an assignment over variables S_X. It holds that $\theta_X[\alpha_{S_X}]$ is equisatisfiable to $\Phi|^{\exists X}_{\alpha_{S_X}}$.*

Proof. As $\exists X$ is the innermost quantifier, all variables in φ are either bound by $\exists X$ or by some outer quantifier. By definition in Equation 12, the abstraction is $\theta_X = enc(\varphi)$. Note that φ and $enc(\varphi)$ are identical up to subformulas ψ of φ with only outer influence ($\psi = \psi^<$), where ψ is replaced in $enc(\varphi)$ by a satisfaction variable. Let α_{S_X} be some assignment over satisfaction variables S_X. By the definition of $enc(\varphi)$ (Equation 13), replacing s_i with $\alpha_{S_X}(s_i)$ leads to formulas which are equal to $\mathbf{T}$ if ψ_i is disjunctive and $\alpha_{S_X}(s_i) = \mathbf{T}$, and to $\mathbf{F}$ if ψ_i is conjunctive and $\alpha_{S_X}(s_i) = \mathbf{F}$. Otherwise, the variable s_i is just removed from the encoded formula $enc(\varphi)$. This matches the definition of the subformula valuation function β_φ resulting from α_{S_X}, thus,

$$\Phi|^{\exists X}_{\alpha_{S_X}} = \Phi|^{\exists X}_{\beta_\varphi} = enc(\varphi)[\alpha_{S_X}] = \theta_X[\alpha_{S_X}] \ ,$$

which we show by structural induction over φ. Let β_φ be the subformula valuation corresponding to α_{S_X}. We show that $\Phi|^{\exists X}_{\beta_\varphi}$ is equal to $\theta_X[\alpha_{S_X}]$. Let ψ_i be an arbitrary non-literal subformula of φ. Further, let ψ_j be an arbitrary direct subformula $\psi_j \in dsf(\psi_i)$. We perform a case distinction on ψ_j:

- Let $\psi_j = \psi_j^=$, thus, ψ_j contains only variables X. The encoding of ψ_j is equal in both cases, as $enc_{\psi_i}(\psi_j) = \psi_j$ and $\beta_\varphi(\psi_j) = \bot$.
- Let $\psi_j = \psi_j^<$, thus, ψ_j contains only variables bound at outer quantifiers. Thus, $enc_{\psi_i}(\psi_j) = s_i$ and the subformula is replaced by a constant in $\Phi|^{\exists X}_{\beta_\varphi}$. Since we replace s_i with $\alpha_{S_X}(s_i)$, we do a further case distinction on $type(\psi_i)$ and $\alpha_{S_X}(s_i)$.
  - Assume $type(\psi_i) = \wedge$ and $\alpha_{S_X}(s_i) = \mathbf{T}$, thus, assigning s_i positively in $enc(\psi_i)$ has the same effect as removing ψ_j.
  - Assume $type(\psi_i) = \vee$ and $\alpha_{S_X}(s_i) = \mathbf{F}$, thus, assigning s_i negatively in $enc(\psi_i)$ has the same effect as removing ψ_j.

– Assume $\mathit{type}(\psi_i) = \wedge$ and $\alpha_{S_X}(s_i) = \mathbf{F}$, thus, assigning s_i negatively in $\mathit{enc}(\psi_i)$ makes $\mathit{enc}(\psi_i)$ unsatisfiable. By definition in Equation 16, $\beta_\varphi(\psi_i) = \mathbf{F}$ as well.

– Assume $\mathit{type}(\psi_i) = \vee$ and $\alpha_{S_X}(s_i) = \mathbf{T}$, thus, assigning s_i positively in $\mathit{enc}(\psi_i)$ makes $\mathit{enc}(\psi_i)$ valid. By definition in Equation 16, $\beta_\varphi(\psi_i) = \mathbf{T}$ as well.

If neither of the base cases above applies, the claim follows by induction. □

The following two lemmata formalize the duality of the abstraction. These statements are used to argue over the dual abstraction. The former states that the abstraction is dual with respect to negation of the formula except for the satisfaction variables. It shows that the dual abstraction is unsatisfiable when assuming a satisfying assignment of the abstraction. The latter lemma shows the correctness of the dual propagation for the innermost quantifier.

Lemma 12 (Duality of θ_X)**.** *Let Φ be a QBF with propositional formula φ, let $\exists X$ be a quantifier of Φ, and let α_X and α_{S_X} be assignments. It holds that*

$$\mathit{enc}(\psi_i)[\alpha_X \mathbin{\dot\sqcup} \alpha_{S_X}] \leftrightarrow \neg \mathit{enc}(\overline{\psi_i})[\alpha_X \mathbin{\dot\sqcup} \overline{\alpha}_{S_X}] \ .$$

Proof. As α_{S_X} abstracts the outer assignments as subformula valuations, α_{S_X} needs to be negated to represent the same assignments in $\overline{\theta}_X$. By structural induction, it is straightforward to show that $\mathit{enc}(\psi_i)$ and $\mathit{enc}(\overline{\psi_i})$ are dual with the exception of variables from S: A disjunction in $\mathit{enc}(\psi_i)$ is a conjunction in $\mathit{enc}(\overline{\psi_i})$ and vice versa, a literal l of quantifier $\exists X$ in $\mathit{enc}(\psi_i)$ is negated $\neg l$ in $\mathit{enc}(\overline{\psi_i})$. Only satisfaction variables appear positively in both formulas. □

Lemma 13. *Let Φ be a QBF with propositional formula φ, let $\exists X$ be the innermost quantifier, and let α_X and α_{S_X} be satisfying assignments of θ_X. It holds that $\overline{\theta}_X[\alpha_X \mathbin{\dot\sqcup} \overline{\alpha}_{S_X}]$ is unsatisfiable. Let β be some set of failed assumptions, that is, $\beta \sqsubseteq \alpha_X \mathbin{\dot\sqcup} \overline{\alpha}_{S_X}$ and $\overline{\theta}_X[\beta]$ is unsatisfiable. Then, $\overline{\beta}|_{S_X} \sqsubseteq \alpha^+_{S_X}$ and $\Phi|^{\exists X}_{\overline{\beta}|_{S_X}[\bot\mapsto\mathbf{F}]}$ is true.*

Proof. By the definition of the abstractions, it holds that $\theta_X = \mathit{enc}(\varphi)$ and $\overline{\theta}_X = \mathit{enc}(\overline{\varphi})$. Lemma 12 shows that if $\alpha_X \mathbin{\dot\sqcup} \alpha_{S_X}$ is a satisfying assignment of $\mathit{enc}(\varphi)$, the assignment $\alpha_X \mathbin{\dot\sqcup} \overline{\alpha}_{S_X}$ falsifies $\mathit{enc}(\overline{\varphi})$. By definition of failed assumptions, $\beta \sqsubseteq \alpha_X \mathbin{\dot\sqcup} \overline{\alpha}_{S_X}$, i.e., there is no α with $\beta \sqsubseteq \alpha$ that satisfies $\overline{\theta}_X$, hence, all $\alpha^*_{S_X}$ with $\overline{\beta}|_{S_X} \sqsubseteq \alpha^*_{S_X}$ satisfy $\theta_X[\alpha_X]$. Together with Lemma 11, this shows that $\Phi|^{\exists X}_{\overline{\beta}|_{S_X}[\bot\mapsto\mathbf{F}]}$ is true. □

Correctness of solve-nnf**.** Finally, we are able to prove the correctness of the solve-nnf algorithm. As in the case for CNF, we prove the correctness by induction over the quantifier prefix.

Lemma 14. *Let Φ be a QBF with propositional formula φ, let $\mathcal{Q}X.\,\Psi$ be a quantified subformula of Φ, and let α_{S_X} be an assignment of the satisfaction variables S_X.*

- *If $\Phi|^{\mathcal{Q}X}_{\alpha_{S_X}}$ is winning for $\mathcal{Q}X$, then* solve-nnf*($\mathcal{Q}X$, Ψ, α_{S_X}) returns $\mathsf{Sat}_{\mathcal{Q}}(\beta_{S_X})$ where $\beta_{S_X} \sqsubseteq \alpha^+_{S_X}$ and $\Phi|^{\mathcal{Q}X}_{\beta_{S_X}[\bot\mapsto\mathbf{F}]}$ is winning for $\mathcal{Q}X$.*

- *If $\Phi|^{\mathcal{Q}X}_{\alpha_{S_X}}$ is losing for $\mathcal{Q}X$, then* SOLVE-NNF*($\mathcal{Q}X$, Ψ, α_{S_X}) returns* $\mathsf{Unsat}_{\mathcal{Q}}(\beta_{S_X})$ *where* $\beta_{S_X} \sqsubseteq \alpha^-_{S_X}$ *and* $\Phi|^{\mathcal{Q}X}_{\beta_{S_X}[\bot \mapsto \mathbf{T}]}$ *is losing for $\mathcal{Q}X$.*

Proof. We prove the statement by structural induction over the quantifier prefix. For this proof, we can restrict $\mathcal{Q}$ to $\exists$ as the universal case is completely dual (Lemma 8). The base case $\exists X$ distinguishes whether $\Phi|^{\exists X}_{\alpha_{S_X}}$ is true or false. In both cases, we use the equisatisfiability of $\Phi|^{\exists X}_{\alpha_{S_X}}$ and $\theta_X[\alpha_{S_X}]$ (Lemma 11). In case the formula is true, we additionally have to use the correctness of the dual propagation as established in Lemma 13. In the induction step, i.e., a quantifier alternation $\exists X. \forall Y$, we perform a case distinction on the value of $\Phi|^{\exists X}_{\alpha_{S_X}}$ as well. If it is true, there is a satisfying assignment α_X such that $\Phi|^{\exists X}_{\alpha_{S_X}}[\alpha_X]$ is losing for $\forall Y$. Applying induction hypothesis and dual propagation gives the required witness. In case the abstraction produces falsifying assignments, the subsequent refinement excludes them from the abstraction, hence, eventually a satisfying assignment is reached. If $\Phi|^{\exists X}_{\alpha_{S_X}}$ is false, every assignment α_X is winning for $\forall Y$ and, thus, leads to a refinement of the abstraction θ_X. The abstraction becomes eventually unsatisfiable (under the assignment α_{S_X}) and the failed assumption represents the required witness. The detailed proof follows.

Induction Base. Let $\exists X. \varphi$ be the innermost quantifier of Φ and let α_{S_X} be some assignment over S_X. We distinguish whether $\Phi|^{\exists X}_{\alpha_{S_X}}$ is true or false:

- Assume that $\Phi|^{\exists X}_{\alpha_{S_X}}$ is true. By Lemma 11, the truth of $\Phi|^{\exists X}_{\alpha_{S_X}}$ witnesses the satisfiability of $\theta_X[\alpha_{S_X}]$. By Lemma 13, the return value of SOLVE-NNF in line 10 meets the requirements.

- Assume that $\Phi|^{\exists X}_{\alpha_{S_X}}$ is false. By Lemma 11 it follows that $\theta_X[\alpha_{S_X}]$ is unsatisfiable. Thus, the algorithm returns $\mathsf{Unsat}(\beta_{S_X})$ where β_{S_X} are the failed assumptions of SAT(θ_X, α_{S_X}) which implies that $\beta_{S_X} \sqsubseteq \alpha^-_{S_X}$. As $\theta_X[\beta_{S_X}[\bot \mapsto \mathbf{T}]]$ is unsatisfiable, $\Phi|^{\exists X}_{\beta_{S_X}[\bot \mapsto \mathbf{T}]}$ is false by Lemma 11, thus, β meets the requirements.

The base case for universal formulas $\forall X. \varphi$ follows from the existential cases by Lemma 8.

Induction Step. Let $\exists X. \forall Y$ be a quantifier alternation of Φ and let α_{S_X} be some assignment over S_X. We distinguish whether $\Phi|^{\exists X}_{\alpha_{S_X}}$ is true or false:

- Assume that $\Phi|^{\exists X}_{\alpha_{S_X}}$ is true. Thus, there is a satisfying assignment α_X for the variables X such that $(\Phi|^{\exists X}_{\alpha_{S_X}})[\alpha_X]$ is true. We define the "optimal" assignment of the assumption variables $\alpha^*_{A_X}$, that is, the minimal assignment with respect to the number of assumptions $(\alpha^*_A(a_i) = \mathbf{T})$ for the given assignment α_X, as

$$\alpha^*_{A_X}(a_i) = \begin{cases} \mathbf{F} & \text{if } \alpha_X \mathbin{\dot\sqcup} \alpha_{S_X} \vDash \mathit{enc}(\psi_i) \\ \mathbf{T} & \text{otherwise} \end{cases} \quad .$$

The definition of the abstraction θ_X (Equation 12) is

$$\theta_X = \bigwedge_{a_i \in A_X} a_i \vee \mathit{enc}(\psi_i) \enspace .$$

The combined assignment $\alpha_X \dot\sqcup \alpha^*_{A_X}$ is, thus, a satisfying assignment of the abstraction $\theta_X[\alpha_{S_X}]$ initially. We perform a case distinction on the returned assignment of the SAT solver in line 3.

- We assume that the SAT call in line 3 returns $\alpha_X \dot\sqcup \alpha^*_{A_X}$. Let $\alpha^*_{S_Y}$ be the assignment constructed in line 5. By Lemma 9, it holds that $(\Phi|^{\exists X}_{\alpha_{S_X}})[\alpha_X] = \Phi|^{\forall Y}_{\alpha^*_{S_Y}}$ is true and, thus, losing for $\forall Y$. By induction hypothesis we deduce that SOLVE-NNF($\forall Y. \Psi$, $\alpha^*_{S_Y}$) returns $\mathsf{Sat}(\beta_{S_Y})$ with $\beta_{S_Y} \sqsubseteq \alpha^-_{S_Y}$ where $\Phi|^{\forall Y}_{\beta_{S_Y}[\bot\mapsto 1]}$ is true. Subsequently, the dual abstraction $\overline{\theta}_X$ is refined (line 7) and SOLVE-NNF returns $\mathsf{Sat}(\beta_{S_X})$ where $\beta_{S_X} = \textsc{optimize}(\alpha_X, \alpha_{S_X})$ (line 8).
  It remains to show that $\Phi|^{\exists X}_{\beta_{S_X}[\bot\mapsto\mathbf{F}]}$ is true and $\beta_{S_X} \sqsubseteq \alpha^+_{S_X}$. First, we show that $\overline{\theta}_X[\alpha_X \dot\sqcup \overline{\alpha}_{S_X}]$ is unsatisfiable. Initially, the dual abstraction is defined as
  $$\overline{\theta}_X = \bigwedge_{a_i \in A_X} a_i \vee enc(\overline{\psi_i}) \enspace .$$
  The refinement clause for the dual abstraction is $\xi \coloneqq \bigvee_{s_i \in \beta^0_{S_Y}} \overline{a_i}$ (line 7). As established by Lemma 12, for every $a_i \in A_X$ it holds that $enc(\psi_i)[\alpha_X \dot\sqcup \alpha_{S_X}] \leftrightarrow \neg enc(\overline{\psi_i})[\alpha_X \dot\sqcup \overline{\alpha}_{S_X}]$. By the definition of θ_X, for every $a_i \in \alpha^0_{A_X}$ it holds that $enc(\psi_i)[\alpha_X \dot\sqcup \alpha_{S_X}] = \mathbf{T}$. As $\alpha_{A_X}(a_i) = \alpha_{S_Y}(s_i)$ for every $a_i \in A_X$ and $\beta_{S_Y} \sqsubseteq \alpha^-_{S_Y}$ it follows that $enc(\overline{\psi_i})[\alpha_X \dot\sqcup \overline{\alpha}_{S_X}] = \mathbf{F}$ for every $s_i \in \beta^0_{S_Y}$. This shows that $\overline{\theta}_X[\alpha_X \dot\sqcup \overline{\alpha}_{S_X}]$ is unsatisfiable after the refinement ξ. Let β be the failed assumptions. The returned assignment is $\beta_{S_X} = \overline{\beta}|_{S_X}$, thus $\beta_{S_X} \sqsubseteq \alpha^+_{S_X}$. For every α'_{S_X} with $\beta_{S_X} \sqsubseteq \alpha'_{S_X}$ it holds that $\overline{\theta}_X[\alpha_X \dot\sqcup \overline{\alpha}'_{S_X}]$ is unsatisfiable as it falsifies the refinement ξ. Thus, one can define a corresponding optimal α'_{A_X} that satisfies θ_X and for the resulting α'_{S_Y} it holds that $\Phi|^{\forall Y}_{\alpha'_{S_Y}}$ is true as $\beta_{S_Y} \sqsubseteq {\alpha'}^-_{S_Y}$. Hence, $\Phi|^{\exists X}_{\beta_{S_X}[\bot\mapsto\mathbf{F}]}$ is true.
- Assume that the SAT call in line 3 returns a different assumption α'_{A_X}. Either α'_{A_X} corresponds to α_X and is non-minimal, i.e., ${\alpha^*_{A_X}}^+ \sqsubseteq {\alpha'_{A_X}}^+$, or it corresponds to a different assignment α'_X. The call to SOLVE-NNF may either return Sat or a counterexample $\mathsf{Unsat}(\beta_{S_Y})$ with $\beta_{S_Y} \sqsubseteq \alpha^+_{S_Y}$. We consider the latter case as in the former case SOLVE-NNF also returns Sat and the same argumentation as in the previous case applies.
  The subsequent refinement in line 9 requires that one of the not satisfied subformulas ψ_i with $\beta_{S_Y}(s_i) = \alpha_{A_X}(a_i) = \mathbf{T}$ has to be satisfied in the next iteration and the corresponding refinement clause is $\xi \coloneqq \bigvee_{s_i \in \beta^1_{S_Y}} \overline{a_i}$. By construction of $\alpha^*_{A_X}$ as the minimal assignment corresponding to α_X, $\alpha^*_{A_X} \not\models \xi$ contradicts that α_X is a satisfying assignment of $\Phi|^{\exists X}_{\alpha_{S_X}}$. Hence, $\alpha_X \dot\sqcup \alpha^*_{A_X}$ is still a satisfying assignment for the refined abstraction $\theta'_X[\alpha_{S_X}]$. The refinement also reduces the number of A_X assignments by at least 1 and, thus, brings us one step closer to a satisfying assignment.

- Assume that $\Phi|^{\exists X}_{\alpha_{S_X}}$ is false. For every assignment α_X, it holds that $(\Phi|^{\exists X}_{\alpha_{S_X}})[\alpha_X]$ is false. The abstraction θ_X is initially satisfiable for every choice of α_{S_X} (every a_i can be set to true, see Equation 12). Let α be a such satisfying assignment of $\theta_X[\alpha_{S_X}]$. We define $\alpha_X \coloneqq \alpha|_X$ and $\alpha_{A_X} \coloneqq \alpha|_{A_X}$. By construction of θ_X (Equation 12), $\alpha_X \dot{\sqcup} \alpha_{S_X} \nvDash enc(\psi_i)$ implies that $\alpha_{A_X}(a_i) = \mathbf{T}$. We define the assignment with optimal assumptions $\alpha^*_{A_X}$ as $\alpha^*_{A_X}(a_i) = \mathbf{F}$ if, and only if, $\alpha_X \sqcup \alpha_{S_X} \vDash enc(\psi_i)$. Note that $\alpha_X \dot{\sqcup} \alpha^*_{A_X}$ is a satisfying assignment of $\theta_X[\alpha_{S_X}]$. We show that even with optimal assumptions $\alpha^*_{A_X}$, the quantified subformula is unsatisfiable and the subsequent refinement step excludes assignment α_{A_X} from the abstraction θ_X.

Let α'_{S_Y} and $\alpha^*_{S_Y}$ be the assignments after line 5 with respect to α_{A_X} and $\alpha^*_{A_X}$, respectively. From the construction, we know that ${\alpha^*_{A_X}}^+ \sqsubseteq {\alpha_{A_X}}^+$, by the optimality of α^*_A, and thereby ${\alpha^*_{S_Y}}^+ \sqsubseteq {\alpha'_{S_Y}}^+$. By Lemma 9, it holds that $(\Phi|^{\exists X}_{\alpha_{S_X}})[\alpha_X]$ and $\Phi|^{\forall Y}_{\alpha^*_{S_X}}$ are equisatisfiable and, thus, winning for $\forall$. By the monotonicity condition given in Lemma 10, it follows that $\Phi|^{\forall X}_{\alpha'_{S_X}}$ is false as well. By induction hypothesis, SOLVE-NNF($\forall Y$, Ψ, α'_{S_X}) returns $\mathsf{Unsat}(\beta_{S_Y})$ such that $\beta_{S_Y} \sqsubseteq {\alpha'_{S_Y}}^+$ and $\Phi|^{\forall}_{\beta_{S_Y}[\bot\mapsto\mathbf{F}]}$ is false. As $\beta^1_{S_Y} \subseteq {\alpha'_{S_Y}}^1 = \{a_i \in A_X \mid \alpha_A(a_i) = \mathbf{T}\}$, the following refinement with clause $\bigvee_{s_i\in\beta^1_S} \overline{a}_i$ excludes assignment α_{A_X} from θ_X. As there are only finitely many refinement clauses, the SAT call in line 3 eventually becomes unsatisfiable when assuming α_{S_X}. Let θ'_X be the abstraction at this point and let β'_{S_X} be the failed assumptions, i.e., $\beta'_{S_X} \sqsubseteq \alpha^+_{S_X}$.

Let $\alpha''_{S_X} = \beta'_{S_X}[\bot \mapsto \mathbf{T}]$. It remains to show that $\Phi|^{\exists X}_{\alpha''_{S_X}}$ is false. Assume for contradiction that there is some α_X such that $(\Phi|^{\exists}_{\alpha''_{S_X}})[\alpha_X]$ is true. It holds that $\theta'_X[\alpha_X \dot{\sqcup} \alpha''_{S_X}]$ is unsatisfiable, whereas $\theta_X[\alpha_X \dot{\sqcup} \alpha''_{S_X}]$ is satisfiable. Thus, the assignment α_X was excluded due to refinements. Let α''_{A_X} be the optimal assumption assignment corresponding to α_X. As the refinement only excludes A_X assignments corresponding to some S_Y assignment β''_{S_Y} such that $\Phi|^{\forall}_{\beta''_{S_Y}[\bot\mapsto\mathbf{F}]}$ is false, which contradicts our assumption.

The induction step for quantifier alternation $\forall X.\,\exists Y$ follows from $\exists X.\,\forall Y$ and Lemma 8. $\square$

Since the main algorithm SOLVE directly calls into SOLVE-NNF, the following theorem follows immediately from Lemma 14.

Theorem 5. SOLVE *returns* Sat *if, and only if,* Φ *is true.*

7.3 Optimizations

In this section, we describe optimizations for the algorithm. Compared to CNF, there are less opportunities in the algorithm as the dual abstraction already takes care of generating and translating witnesses.

As shown in the last section, the satisfaction assignments α_S correspond to partial formula evaluations. In the same way as the CNF algorithm, the abstraction only builds an implication $a_i \vee enc(\psi_i)$, thus, assumption assignments α_A may not be optimal. Fix some quantifier $\mathcal{Q}X$. During the execution of the algorithm, we maintain the partial evaluation β_φ of φ under the current variable assignment α_V of variables bound at X or at some outer quantifier and we use this evaluation to build optimal assignments. If $\beta_\varphi(\psi_i) = \mathbf{T}$ for some $a_i \in A_X$, then we set $\alpha_{A_X}(a_i)$ to $\mathbf{F}$.

Lastly, and already noted by other NNF approaches [44], subformulas $\psi \in \mathit{sf}(\varphi)$ do not need to be in negation normal form if ψ is only influenced by variables of a single quantifier, that is, $\psi = \psi^{=}$. For example, the following formula $\forall x.\, \exists y, z.\, x \wedge (y \leftrightarrow z)$ can be solved with the algorithm presented above without modifications.

7.4 Function Extraction

The overall approach for function extraction algorithm is the same as the one described in Section 5. For every quantifier $\exists X$, we store a sequence of pairs $\langle \beta_{S_X}, \alpha_X \rangle \in (\mathcal{A}_\perp(S_X) \times \mathcal{A}(X))$ and these pairs can be obtained from the algorithm by the returned value β_{S_X} after the dual abstraction optimization (lines 8 and 10). Next, we define the reverse function of the abstraction $\mathit{inv}_{\mathcal{Q}X} \colon \mathcal{A}_\perp(S_X) \to \mathcal{B}(V)$ that maps an assignment β_{S_X} to a propositional formula over variables V bound by outer quantifiers (with respect to X). Intuitively, $\mathit{inv}_{\mathcal{Q}X}(\beta_{S_X})$ describes those assignments that lead to β_{S_X} in the abstraction of quantifier $\mathcal{Q}X$. We define $\mathit{inv}_{\mathcal{Q}X}$ as

$$\mathit{inv}_{\exists X}(\beta_{S_X}) \coloneqq \bigwedge_{s_i \in \beta^1_{S_X}} \mathit{outer}(\psi_i) \tag{17}$$

where *outer* is defined as

$$\mathit{outer}(\psi_i) = \begin{cases} \displaystyle\bigwedge_{\substack{\psi_j \in \mathit{dsf}(\psi_i) \\ \psi_j^{<} = \psi_j}} \psi_j & \text{if } \mathit{type}(\psi_i) = \wedge \\ \displaystyle\bigvee_{\substack{\psi_j \in \mathit{dsf}(\psi_i) \\ \psi_j^{<} = \psi_j}} \psi_j & \text{otherwise} \end{cases}$$

The definition of the extracted function f_x for some $x \in X$ follows then by Equation 10.

Example 9. We show the function extraction for our running example

$$\exists x.\, \forall v, w.\, \exists y.\, \underbrace{(x \vee v \vee \overbrace{(y \wedge w)}^{\psi_3})}_{\psi_2} \wedge \underbrace{(\overline{x} \vee \overbrace{(v \wedge \overline{w})}^{\psi_5} \vee y)}_{\psi_4} \wedge \underbrace{(\overline{v} \vee w \vee \overline{y})}_{\psi_6}$$

From the execution shown in Example 8, we extract the sequences $\langle \emptyset, x \rangle$ and $\langle s_2 s_6, y \rangle \langle s_2 s_4, \overline{y} \rangle$ as described above. The Skolem function for x is the constant $x = \mathbf{T}$. Applying the definition of $\mathit{inv}_{\exists y}$, we get

$$\begin{aligned} \mathit{inv}_{\exists y}(s_2 s_6) &= (x \vee v) \wedge (\overline{v} \vee w) \text{ and} \\ \mathit{inv}_{\exists y}(s_2 s_4) &= (x \vee v) \wedge (\overline{x} \vee (v \wedge \overline{w})) \ . \end{aligned}$$

Thus, the Skolem function f_y is defined as

$$f_y(v, w) = \mathit{inv}_{\exists y}(s_2 s_6)[x \mapsto \mathbf{T}] = (x \vee v) \wedge (\overline{v} \vee w)[x \mapsto \mathbf{T}] = (\overline{v} \vee w) \ .$$

f_x and f_y depend solely on its dependencies and are functionally correct as $\varphi[f_{\{x,y\}}] = ((v \wedge \overline{w}) \vee \overline{v} \vee w)(\overline{v} \vee w \vee (v \wedge \overline{w}))$ is a tautology.

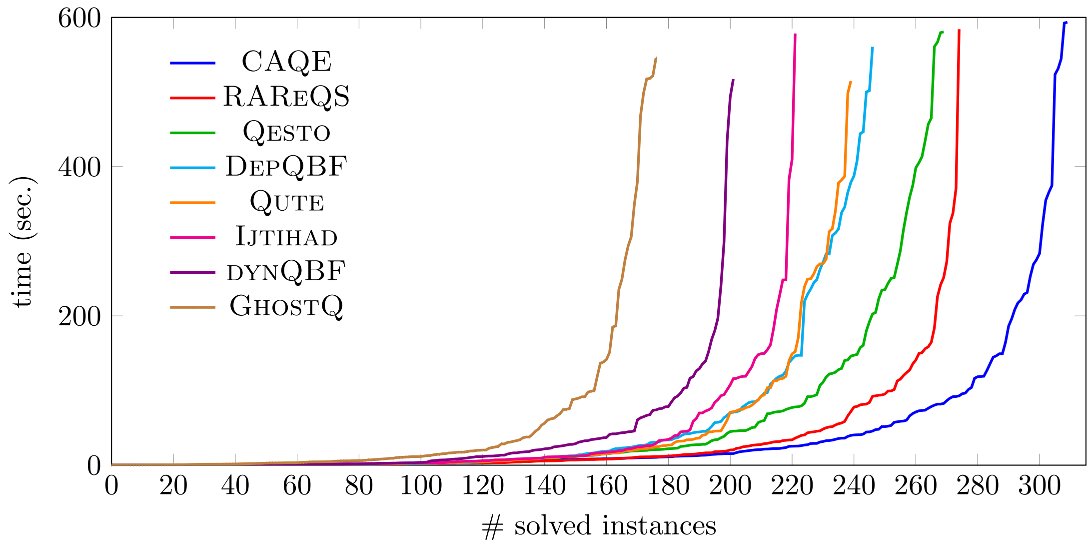


Figure 5: Cactus plot showing the number of solved instances on the prenex CNF benchmark set of QBFEVAL'18 using HQSPre as preprocessor.

8. Experimental Evaluation

For our experiments, we used a machine with a 3.6 GHz quad-core Intel Xeon (E3-1271 v3) processor and 32 GB of memory. The timeout and memout were set to 10 minutes and 8 GB, respectively.

8.1 CAQE

We implemented the clausal abstraction algorithm in a tool called CAQE[5.] (Clausal Abstraction for Quantifier Elimination) that takes as input a quantified Boolean formula encoded in the QDIMACS format. As the solver for the propositional abstractions, we used the SAT solver CryptoMiniSat [69] version 5.0.1. We compare CAQE against publicly available QBF solvers that support the QDIMACS format, namely DepQBF [56] version 6.03, dynQBF [19] version 1.1.1, GhostQ [50] version 2017, Qesto [46] version 1.0, Qute [62] version 1.1, and RAReQS [44] version 1.1. We use the prenex CNF benchmark set from the QBF competition QBFEVAL'18 [6.]. As preprocessors, we used Bloqqer [12] version 031, HQSPre [75] version 1.4, and QRATPre+ [57] version 1.0. The cactus plot given in Figure 5 shows the number of solved instances for the best combination of preprocessor and solver. Detailed solving results are shown in Table 1. CAQE solves overall most instances, followed by RAReQS and Qesto. Further, all solvers solved significantly more instances when using HQSPre compared to Bloqqer. At the same time, the improvement due to HQSPre is much smaller for the solvers CAQE and RAReQS that are based on (partial) expansion then for the other solvers, possibly due to the more aggressive in expansion of universal variables in HQSPre compared to Bloqqer.

5. Source code available at https://github.com/ltentrup/caqe
6. Available at http://www.qbflib.org/qbfeval18.php

Table 1: Number of solved formulas by combinations of solvers and preprocessors on the prenex-CNF benchmark set of QBFEVAL'18. For every combination, we give the number of solved instances overall and broken down by result, that is, satisfiable and unsatisfiable.

preprocessor	HQSPre			Bloqqer			QRATPre+			none		
solver	solved	sat	unsat	solved	sat	unsat	solved	sat	unsat	solved	sat	unsat
CAQE	**309**	**122**	**187**	**273**	**115**	**158**	**161**	63	**98**	141	43	98
RAReQS	274	102	172	247	94	153	136	47	89	139	28	**111**
Qesto	269	108	161	196	89	107	127	52	75	98	29	69
DepQBF	246	97	149	181	91	90	138	**70**	68	136	53	83
Qute	239	79	160	159	58	101	116	40	76	94	17	77
Ijtihad	201	75	146	198	74	124	125	39	86	131	23	108
dynQBF	201	85	116	113	59	54	81	56	25	59	39	20
GhostQ	–	–	–	–	–	–	–	–	–	**176**	**89**	87

Extended Refinements. We discuss the effect of the stronger refinements given in Section 4.3 and the expansion refinement given in Section 6. There is a tradeoff between the *precision* of the abstraction and the cost of these satisfiability calls. The more precise an abstraction, the more losing assignments are excluded, i.e., a higher precision can potentially reduce the number of propositional satisfiability calls. Both presented optimizations can potentially improve the precision, but both of them also may increase the time spent inside the SAT solver. Further, the relative performance of the optimizations depend on the benchmark set as well as the preprocessor that is used, thus, it is advisable to evaluate those optimizations in practice on a case-by-case basis. However, in our experiments, we found that the expansion refinement optimization vastly improves the number of solved instances independently of the preprocessor. Also, when comparing the running times directly, as done in the scatter plot depicted in Figure 6, the negative effect of the running time of the propositional SAT solver is reasonably small.

Regarding the stronger refinements, we found that the effect on instances preprocessed with HQSPre is negligible. When using Bloqqer, however, the optimization improved the number of solved instances significantly. Further, the combination of both refinements, which we call extended refinement (which is also the default configuration used in the evaluation above), is the best performing variant of CAQE when using Bloqqer as preprocessor. In our experiments, the combination performed better than any of the two refinements alone, indicating that they are in some sense orthogonal, as shown in the scatter plots in Figure 6.

Algorithmic Choices. In the following, we want to quantify the impact of the algorithmic choices described in the article. For this setup, we used a version of CAQE which is close to the initial version of Section 4. Then, we enabled one of the algorithmic improvements mentioned in this article to evaluate their impact. The results are given in Table 2. The most impact in terms of additionally solved instances has the expansion refinement which can be explained by the corresponding improvement of the underlying proof system [72]. The sum of additionally solved instances of the optimizations that are enabled by

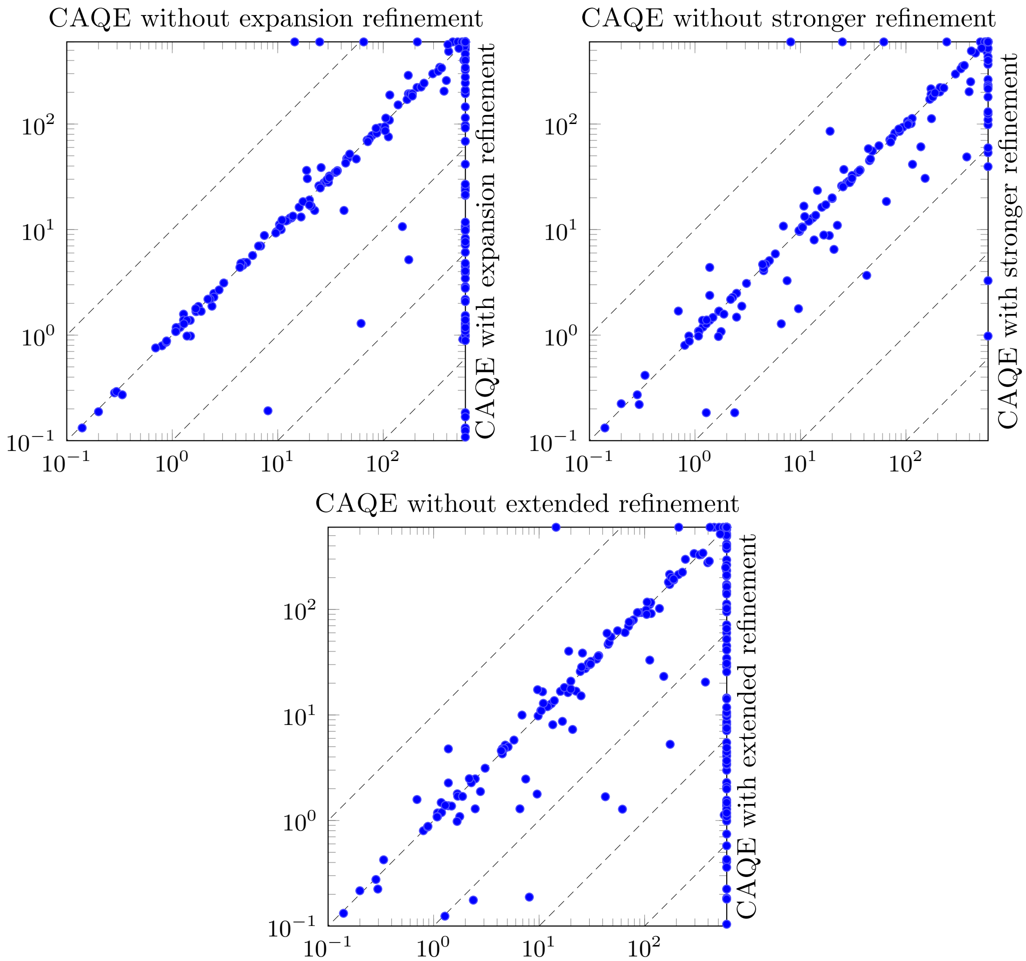


Figure 6: Scatter plot comparing the solving time (in sec.) of CAQE with and without extended refinements (expansion refinement and stronger refinement) and preprocessing using Bloqqer. Both axes have logarithmic scale.

default (304) is smaller than the number of instances solved by CAQE (309) which hints at a positive synergy regarding the combination of individual optimizations.

8.2 QuAbS

We implemented the abstraction algorithm for negation normal form formulas in a solver called QuAbS[7.] (Quantified Abstraction Solver) that takes as input a quantified Boolean formula encoded in the quantified circuit (QCIR) [64] format. As the solver for the propositional abstractions, we used the SAT solver CryptoMiniSat [69] version 5.0.1. We compare QuAbS against the publicly available QBF solvers that support the QCIR format, namely GhostQ [50] version 2017, QFUN [42] version 2018, cQESTO [41] version 2018, and Qute [62] version 1.1. We use the prenex non-CNF benchmark set from the QBF com-

7. Source code available at https://github.com/ltentrup/quabs

Table 2: This table shows the impact of select algorithmic choices on a baseline version of CAQE using HQSPre as preprocessor. The baseline solves 229 instances on the prenex-CNF benchmark set of QBFEVAL'18. For every algorithmic choice, we give the difference of solved instances (Δ) compared to the baseline and detailed results (+) and (−).

Algorithmic choice	default	described in	Δ	+	−
Expansion refinement	yes	Section 6	+50	58	8
Tree-shaped quantifier prefix	yes	Section 4.3	+13	16	3
Stronger refinement	yes	Section 4.3	+6	7	1
Sharing of abstraction literals	yes	Section 4.3	+6	14	8
Equivalence constraints in abstraction	no	[46]	+3	5	2
Backtracking over multiple quantifiers	no	Section 4.3	−1	1	2
Dropping redundant refinement literals	no	Section 4.3	−1	6	7

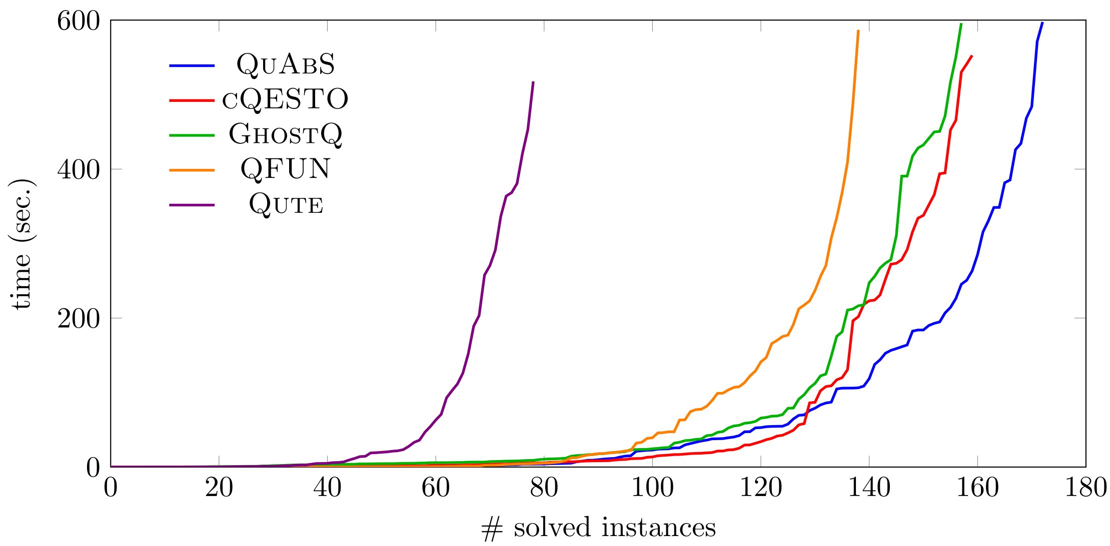


Figure 7: Cactus plot showing the number of solved instances on the QBFEVAL'18 benchmark set.

petition QBFEVAL'18. The results are shown in Figure 7. Despite being slower initially compared to cQESTO, QuAbS solves more instances overall.

Function Extraction. Enabling function extraction outputs a representation of the Skolem and Herbrand function, encoded as And-Inverter-Graph, after determining that the formula is satisfiable and unsatisfiable, respectively. In contrast to CNF solvers, we do not need to disable optimizations [61] nor preprocessing (as we do not use external preprocessors and QuAbS uses only constant propagation as preprocessing technique). Thus, the impact of function extraction is small, as shown by Figure 8 which compares the running time of QuAbS with and without function extraction.

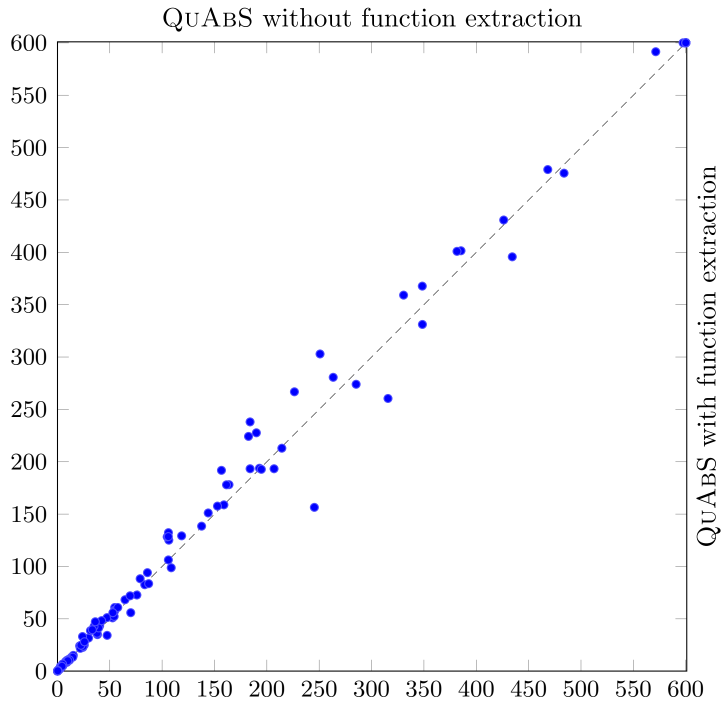


Figure 8: Scatter plot comparing the solving time (in sec.) of QuAbS with and without function extraction.

This makes QuAbS an ideal candidate for applications where solving witnesses are needed: QuAbS is used in the reactive synthesis tool BoSy [24], which won the synthesis track in the reactive synthesis competition (SYNTCOMP) 2016 and 2017 [39, 40]. Further, it is also part of the Petri game solver Adam [26] and the HyperLTL satisfiability solver MGHyper [27].

9. Conclusion

We presented a detailed description and analysis of the clausal abstraction approach—a versatile and performant solving approach for quantified Boolean formulas. A key aspect for the algorithm is the abstraction itself: it can be efficiently implemented using a modern SAT solver and it is flexible, for example, we showed that we can integrate partial expansion in addition to clausal abstraction refinements. On the algorithmic side, the approach of communicating subformula valuations scales from formulas in prenex conjunctive normal form, to non-prenex and negation normal form, respectively, as well as dependency quantified Boolean formulas (DQBF) [73].

Acknowledgments

I thank Bernd Finkbeiner, Jesko Hecking-Harbusch, Markus Rabe, and Martina Seidl for their feedback on earlier versions of this article. I also thank the anonymous reviewers for their comments.

References

[1] Abdelwaheb Ayari and David A. Basin. QUBOS: deciding quantified Boolean logic using propositional satisfiability solvers. In *Proceedings of FMCAD*, **2517** of *LNCS*, pages 187–201. Springer, 2002.

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

[3] Valeriy Balabanov, Jie-Hong Roland Jiang, Mikolas Janota, and Magdalena Widl. Efficient extraction of QBF (counter)models from long-distance resolution proofs. In *Proceedings of AAAI*, pages 3694–3701. AAAI Press, 2015.

[4] Valeriy Balabanov, Jie-Hong Roland Jiang, Christoph Scholl, Alan Mishchenko, and Robert K. Brayton. 2QBF: Challenges and solutions. In *Proceedings of SAT*, **9710** of *LNCS*, pages 453–469. Springer, 2016.

[5] Valeriy Balabanov, Shuo-Ren Lin, and Jie-Hong R. Jiang. Flexibility and optimization of QBF skolem-herbrand certificates. *IEEE Trans. on CAD of Integrated Circuits and Systems*, **35**(9):1557–1568, 2016.

[6] Marco Benedetti. Evaluating QBFs via symbolic skolemization. In *Proceedings of LPAR*, **3452** of *LNCS*, pages 285–300. Springer, 2004.

[7] Marco Benedetti. Extracting certificates from quantified Boolean formulas. In *Proceedings of IJCAI*, pages 47–53. Professional Book Center, 2005.

[8] Marco Benedetti. sKizzo: A suite to evaluate and certify QBFs. In *Proceedings of CADE-20*, **3632** of *LNCS*, pages 369–376. Springer, 2005.

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

[10] Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In *Proceedings of STACS*, **30** of *LIPIcs*, pages 76–89. Schloss Dagstuhl – LZI, 2015.

[11] Armin Biere. Resolve and expand. In *Proceedings of SAT*, **3542** of *LNCS*, pages 59–70. Springer, 2004.

[12] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In *Proceedings of CADE*, **6803** of *LNCS*, pages 101–115. Springer, 2011.

[13] Nikolaj Bjørner and Mikolás Janota. Playing with quantified satisfaction. In *Proceedings of LPAR*, **35** of *EPiC Series in Computing*, pages 15–27. EasyChair, 2015.

[14] Roderick Bloem, Nicolas Braud-Santoni, Vedad Hadzic, Uwe Egly, Florian Lonsing, and Martina Seidl. Expansion-based QBF solving without recursion. In *Proceedings of FMCAD*, pages 1–10. IEEE, 2018.

[15] Roderick Bloem, Uwe Egly, Patrick Klampfl, Robert Könighofer, and Florian Lonsing. SAT-based methods for circuit synthesis. In *Proceedings of FMCAD*, pages 31–34. IEEE, 2014.

[16] Roderick Bloem, Robert Könighofer, and Martina Seidl. SAT-based synthesis methods for safety specs. In *Proceedings of VMCAI*, **8318** of *LNCS*, pages 1–20. Springer, 2014.

[17] Thomas Brihaye, Véronique Bruyère, Laurent Doyen, Marc Ducobu, and Jean-François Raskin. Antichain-based QBF solving. In *Proceedings of ATVA*, **6996** of *LNCS*, pages 183–197. Springer, 2011.

[18] Hans Kleine Büning and Uwe Bubeck. Theory of quantified Boolean formulas. In *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*, pages 735–760. IOS Press, 2009.

[19] Günther Charwat and Stefan Woltran. Dynamic programming-based QBF solving. In *Proceedings of QBF@SAT*, **1719** of *CEUR Workshop Proceedings*, pages 27–40. CEUR-WS.org, 2016.

[20] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of CAV*, **1855** of *LNCS*, pages 154–169. Springer, 2000.

[21] Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *Proceedings of LPAR*, **8312** of *LNCS*, pages 291–308. Springer, 2013.

[22] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in negation normal form. *Constraints*, **14**(1):38–79, 2009.

[23] Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In *Proceedings of TACAS*, **10205** of *LNCS*, pages 354–370, 2017.

[24] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. BoSy: An experimentation framework for bounded synthesis. In *Proceedings of CAV*, **10427** of *LNCS*, pages 325–332. Springer, 2017.

[25] Katalin Fazekas, Marijn J. H. Heule, Martina Seidl, and Armin Biere. Skolem function continuation for quantified Boolean formulas. In *Proceedings of TAP*, **10375** of *LNCS*, pages 129–138. Springer, 2017.

[26] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Symbolic vs. bounded synthesis for Petri games. In *Proceedings of SYNT@CAV*, **260** of *EPTCS*, pages 23–43, 2017.

[27] Bernd Finkbeiner, Christopher Hahn, and Tobias Hans. MGHyper: Checking satisfiability of HyperLTL formulas beyond the $\exists^*\forall^*$ fragment. In *Proceedings of ATVA*, **11138** of *LNCS*, pages 521–527. Springer, 2018.

[28] Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing reactive systems from hyperproperties. In *Proceedings of CAV*, **10981** of *LNCS*, pages 289–306. Springer, 2018.

[29] Bernd Finkbeiner and Leander Tentrup. Fast DQBF refutation. In *Proceedings of SAT*, **8561** of *LNCS*, pages 243–251. Springer, 2014.

[30] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified Boolean formulas. In *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*, pages 761–780. IOS Press, 2009.

[31] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE++: An efficient QBF solver. In *Proceedings of FMCAD*, **3312** of *LNCS*, pages 201–213. Springer, 2004.

[32] Alexandra Goultiaeva and Fahiem Bacchus. Exploiting QBF duality on a circuit representation. In *Proceedings of AAAI*. AAAI Press, 2010.

[33] Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *Proceedings of IJCAI*, pages 546–553. IJCAI/AAAI, 2011.

[34] Alexandra Goultiaeva, Vicki Iverson, and Fahiem Bacchus. Beyond CNF: A circuit-based QBF solver. In *Proceedings of SAT*, **5584** of *LNCS*, pages 412–426. Springer, 2009.

[35] Alexandra Goultiaeva, Martina Seidl, and Armin Biere. Bridging the gap between dual propagation and CNF-based QBF solving. In *Proceedings of DATE*, pages 811–814. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[36] Jesko Hecking-Harbusch and Leander Tentrup. Solving QBF by abstraction. In *Proceedings of GandALF*, **277** of *EPTCS*, pages 88–102, 2018.

[37] Marijn Heule, Martina Seidl, and Armin Biere. A unified proof system for QBF preprocessing. In *Proceedings of IJCAR*, **8562** of *LNCS*, pages 91–106. Springer, 2014.

[38] Marijn J. H. Heule, Martina Seidl, and Armin Biere. Solution validation and extraction for QBF preprocessing. *J. Autom. Reasoning*, **58**(1):97–125, 2017.

[39] 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, and Leander Tentrup. The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. In *Proceedings of SYNT@CAV*, **260** of *EPTCS*, pages 116–143, 2017.

[40] 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, and Adam Walker. The 3rd reactive synthesis competition (SYNTCOMP 2016): Benchmarks, participants & results. In *Proceedings of SYNT@CAV*, **229** of *EPTCS*, pages 149–177, 2016.

[41] Mikolás Janota. Circuit-based search space pruning in QBF. In *Proceedings of SAT*, **10929** of *LNCS*, pages 187–198. Springer, 2018.

[42] Mikolás Janota. Towards generalization in QBF solving via machine learning. In *Proceedings of AAAI*. AAAI Press, 2018.

[43] Mikolás Janota, Radu Grigore, and João Marques-Silva. On QBF proofs and preprocessing. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, **8312** of *LNCS*, pages 473–489. Springer, 2013.

[44] Mikolás Janota, William Klieber, Joao Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, **234**:1–25, 2016.

[45] Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, **577**:25–42, 2015.

[46] Mikolás Janota and Joao Marques-Silva. Solving QBF by clause selection. In *Proceedings of IJCAI*, pages 325–331. AAAI Press, 2015.

[47] Mikolás Janota and João Marques-Silva. An achilles' heel of term-resolution. In *Proceedings of EPIA*, **10423** of *LNCS*, pages 670–680. Springer, 2017.

[48] Toni Jussila, Armin Biere, Carsten Sinz, Daniel Kröning, and Christoph M. Wintersteiger. A first step towards a unified proof checker for QBF. In *Proceedings of SAT*, **4501** of *LNCS*, pages 201–214. Springer, 2007.

[49] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, **117**(1):12–18, 1995.

[50] William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *Proceedings of SAT*, **6175** of *LNCS*, pages 128–142. Springer, 2010.

[51] Nian-Ze Lee, Yen-Shi Wang, and Jie-Hong R. Jiang. Solving exist-random quantified stochastic Boolean satisfiability via clause selection. In *Proceedings of IJCAI*, pages 1339–1345. ijcai.org, 2018.

[52] Florian Lonsing, Fahiem Bacchus, Armin Biere, Uwe Egly, and Martina Seidl. Enhancing search-based QBF solving by dynamic blocked clause elimination. In *Proceedings of LPAR*, **9450** of *LNCS*, pages 418–433. Springer, 2015.

[53] Florian Lonsing and Armin Biere. Nenofex: Expanding NNF for QBF solving. In *Proceedings of SAT*, **4996** of *LNCS*, pages 196–210. Springer, 2008.

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

[55] Florian Lonsing and Uwe Egly. Incremental QBF solving by DepQBF. In *Proceedings of ICMS*, **8592** of *LNCS*, pages 307–314. Springer, 2014.

[56] Florian Lonsing and Uwe Egly. DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In *Proceedings of CADE*, **10395** of *LNCS*, pages 371–384. Springer, 2017.

[57] Florian Lonsing and Uwe Egly. QRAT+: generalizing QRAT by a more powerful QBF redundancy property. In *Proceedings of IJCAR*, **10900** of *LNCS*, pages 161–177. Springer, 2018.

[58] Christian Miller, Christoph Scholl, and Bernd Becker. Proving QBF-hardness in bounded model checking for incomplete designs. In *Proceedings of MTV*, pages 23–28. IEEE Computer Society, 2013.

[59] Massimo Narizzano, Luca Pulina, and Armando Tacchella. The QBFEVAL web portal. In *Proceedings of JELIA*, **4160** of *LNCS*, pages 494–497. Springer, 2006.

[60] Aina Niemetz, Mathias Preiner, and Armin Biere. Turbo-charging lemmas on demand with don't care reasoning. In *Proceedings of FMCAD*, pages 179–186. IEEE, 2014.

[61] Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF - (tool presentation). In *Proceedings of SAT*, **7317** of *LNCS*, pages 430–435. Springer, 2012.

[62] Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. In *Proceedings of SAT*, **10491** of *LNCS*, pages 298–313. Springer, 2017.

[63] Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (QBFEVAL'16 and QBFEVAL'17). *Artif. Intell.*, **274**:224–248, 2019.

[64] QBF Gallery 2014. QCIR-G14: A non-prenex non-CNF format for quantified Boolean formulas. Technical report, 2014.

[65] Markus N. Rabe and Sanjit A. Seshia. Incremental determinization. In *Proceedings of SAT*, **9710** of *LNCS*, pages 375–392. Springer, 2016.

[66] Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *Proceedings of FMCAD*, pages 136–143. IEEE, 2015.

[67] Markus N. Rabe, Leander Tentrup, Cameron Rasmussen, and Sanjit A. Seshia. Understanding and extending incremental determinization for 2QBF. In *Proceedings of CAV*, **10982** of *LNCS*, pages 256–274. Springer, 2018.

[68] Christoph Scholl and Florian Pigorsch. The QBF solver AIGSolve. In *Proceedings of QBF@SAT*, **1719** of *CEUR Workshop Proceedings*, pages 55–62. CEUR-WS.org, 2016.

[69] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Proceedings of SAT*, **5584** of *LNCS*, pages 244–257. Springer, 2009.

[70] Igor Stéphan and Benoit Da Mota. A unified framework for certificate and compilation for QBF. In *Proceedings of ICLA*, **5378** of *LNCS*, pages 210–223. Springer, 2009.

[71] Leander Tentrup. Non-prenex QBF solving using abstraction. In *Proceedings of SAT*, **9710** of *LNCS*, pages 393–401. Springer, 2016.

[72] Leander Tentrup. On expansion and resolution in CEGAR based QBF solving. In *Proceedings of CAV*, **10427** of *LNCS*, pages 475–494. Springer, 2017.

[73] Leander Tentrup and Markus N. Rabe. Clausal abstraction for DQBF. In *Proceedings of SAT*, **11628** of *LNCS*, pages 388–405. Springer, 2019.

[74] Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic*, **2**(115-125):10–13, 1968.

[75] Ralf Wimmer, Sven Reimer, Paolo Marin, and Bernd Becker. HQSpre - an effective preprocessor for QBF and DQBF. In *Proceedings of TACAS*, **10205** of *LNCS*, pages 373–390, 2017.

[76] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *Proceedings of ICCAD*, pages 442–449. ACM / IEEE Computer Society, 2002.

[77] Wenhui Zhang. QBF encoding of temporal properties and QBF-based verification. In *Proceedings of IJCAR*, **8562** of *LNCS*, pages 224–239. Springer, 2014.