Title
Understanding and Extending Incremental Determinization for 2QBF

ISBN
978-3-319-96141-5

Authors
Rabe, Markus N
Tentrup, Leander
Rasmussen, Cameron
et al.

Publication Date
2018

DOI
10.1007/978-3-319-96142-2_17

Peer reviewed

Understanding and Extending Incremental Determinization for 2QBF

Markus N. Rabe[1], Leander Tentrup[2], Cameron Rasmussen[1], and Sanjit A. Seshia[1]

[1]University of California, Berkeley
[2]Saarland University

Abstract. Incremental determinization is a recently proposed algorithm for solving quantified Boolean formulas with one quantifier alternation. In this paper, we formalize incremental determinization as a set of inference rules to help understand the design space of similar algorithms. We then present additional inference rules that extend incremental determinization in two ways. The first extension integrates the popular CEGAR principle and the second extension allows us to analyze different cases in isolation. The experimental evaluation demonstrates that the extensions significantly improve the performance.

1 Introduction

Solving quantified Boolean formulas (QBFs) is one of the core challenges in automated reasoning and is particularly important for applications in verification and synthesis. For example, program synthesis with syntax guidance [1, 2] and the synthesis of reactive controllers from LTL specifications has been encoded in QBF [3, 4]. Many of these problems require only formulas with one quantifier alternation (2QBF), which are the focus of this paper.

Algorithms for QBF and program synthesis largely rely on the counterexample-guided inductive synthesis principle (CEGIS) [5], originating in abstraction refinement (CEGAR) [6, 7]. For example, for program synthesis, CEGIS-style algorithms alternate between generating candidate programs and checking them for counter-examples, which allows us to lift arbitrary verification approaches to synthesis algorithms. Unfortunately, this approach often degenerates into a plain guess-and-check loop when counter-examples cannot be generalized effectively. This carries over to the simpler setting of 2QBF. For example, even for a simple formula such as $\forall x.\exists y.\ x = y$, where x and y are 32-bit numbers, most QBF algorithms simply enumerate all 2^{32} pairs of assignments. In fact, even the modern QBF solvers diverge on this formula when preprocessing is deactivated.

Recently, Incremental Determinization (ID) has been suggested to overcome this problem [8]. ID represents a departure from the CEGIS approach in that it is structured around identifying which variables have unique Skolem functions. (To prove the truth of a 2QBF $\forall x.\exists y.\ \varphi$ we have to find Skolem functions f mapping x to y such that $\varphi[f/y]$ is valid.) After assigning Skolem functions to

a few of the existential variables, the propagation procedure determines Skolem functions for other variables that are uniquely implied by that assignment. When the assignment of Skolem functions turns out to be incorrect, ID analyzes the conflict, derives a conflict clause, and backtracks some of the assignements. In other words, ID lifts CDCL to the space of Skolem functions.

ID can solve the simple example given above and shows good performance on various application benchmarks. Yet, the QBF competitions have shown that the relative performance of ID and CEGIS still varies a lot between benchmarks [9]. A third family of QBF solvers, based on the *expansion* of universal variables [10–12], shows yet again different performance characteristics and outperforms both ID and CEGIS on some (few) benchmarks. This variety of performance characteristics of different approaches indicates that current QBF solvers could be significantly improved by integrating the different reasoning principles.

In this paper, we first formalize and generalize ID [8] (Section 3). This helps us to disentangle the working principles of the algorithm from implementation-level design choices. Thereby our analysis of ID enables a systematic and principled search for better algorithms for quantified reasoning. To demonstrate the value and flexibility of the formalization, we present two extensions of ID that integrate CEGIS-style inductive reasoning (Section 4) and expansion (Section 5). In the experimental evaluation we demonstrate that both extensions significantly improve the performance compared to plain ID (Section 6).

Related work. This work is written in the tradition of works such as the Model Evolution Calculus [13], AbstractDPLL [14], MCSAT [15], and recent calculi for QBF [16], which present search algorithms as inference rules to enable the study and extension of these algorithms. ID and the inference rules presented in this paper can be seen as an instantiation of the more general frameworks, such as MCSAT [15] or Abstract Conflict Driven Learning [17].

Like ID, quantified conflict-driven clause learning (QCDCL) lifts CDCL to QBF [18, 19]. The approaches differ in that QCDCL does not reason about functions, but only about values of variables. Fazekas et al. have formalized QCDCL as inference rules [16].

2QBF solvers based on CEGAR/CEGIS search for universal assignments and matching existential assignments using two SAT solvers [5,20,21]. There are several generalizations of this approach to QBF with more than one quantifier alternation [22–26].

2 Preliminaries

Quantified Boolean formulas over a finite set of variables $x \in X$ with domain $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$ are generated by the following grammar:

$$\varphi \coloneqq \mathbf{0} \mid \mathbf{1} \mid x \mid \neg\varphi \mid (\varphi) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi$$

We consider all other logical operations, including implication, XOR, and equality as syntactic sugar with the usual definitions. We abbreviate multiple quantifications $Qx_1.Qx_2.\ldots Qx_n.\varphi$ using the same quantifier $Q \in \{\forall, \exists\}$ by the quantification over the set of variables $X = \{x_1, \ldots, x_n\}$, denoted as $QX.\varphi$.

An *assignment* $\boldsymbol{x}$ to a set of variables X is a function $\boldsymbol{x} : X \to \mathbb{B}$ that maps each variable $x \in X$ to either $\mathbf{1}$ or $\mathbf{0}$. Given a propositional formula φ over variables X and an assignment $\boldsymbol{x}'$ to $X' \subseteq X$, we define $\varphi(\boldsymbol{x}')$ to be the formula obtained by replacing the variables X' by their truth value in $\boldsymbol{x}'$. By $\varphi(\boldsymbol{x}', \boldsymbol{x}'')$ we denote the replacement by multiple assignments for disjoint sets $X', X'' \subseteq X$.

A quantifier $Q\, x.\, \varphi$ for $Q \in \{\exists, \forall\}$ *binds* the variable x in its subformula φ and we assume w.l.o.g. that every variable is bound at most once in any formula. A *closed* QBF is a formula in which all variables are bound. We define the dependency set of an existentially quantified variable y in a formula φ as the set $dep(y)$ of universally quantified variables x such that φ's subformula $\exists y.\, \psi$ is a subformula of φ's subformula $\forall x.\psi'$. A *Skolem function* f_y maps assignments to $dep(y)$ to a truth value. We define the truth of a QBF φ as the existence of Skolem functions $f_Y = \{f_{y_1}, \ldots, f_{y_n}\}$ for the existentially quantified variables $Y = \{y_1, \ldots, y_n\}$, such that $\varphi(\boldsymbol{x}, f_Y(\boldsymbol{x}))$ holds for every $\boldsymbol{x}$, where $f_Y(\boldsymbol{x})$ is the assignment to Y that the Skolem functions f_Y provide for $\boldsymbol{x}$.

A formula is in *prenex normal form*, if the formula is closed and starts with a sequence of quantifiers followed by a propositional subformula. A formula φ is in the kQBF fragment for $k \in \mathbb{N}^+$ if it is closed, in prenex normal form, and has exactly $k-1$ alternations between $\exists$ and $\forall$ quantifiers.

A *literal* l is either a variable $x \in X$, or its negation $\neg x$. Given a set of literals $\{l_1, \ldots, l_n\}$, their disjunction $(l_1 \vee \ldots \vee l_n)$ is called a *clause* and their conjunction $(l_1 \wedge \ldots \wedge l_n)$ is called a *cube*. We use $\bar{l}$ to denote the literal that is the logical negation of l. We denote the variable of a literal by $var(l)$ and lift the notion to clauses $var(l_1 \vee \cdots \vee l_n) = \{var(l_1), \ldots, var(l_n)\}$.

A propositional formula is in conjunctive normal form (CNF), if it is a conjunction of clauses. A prenex QBF is in prenex conjunctive normal form (PCNF) if its propositional subformula is in CNF. Every QBF φ can be transformed into an equivalent PCNF with size $O(|\varphi|)$ [27].

Resolution is a well-known proof rule that allows us to merge two clauses as follows. Given two clauses $C_1 \vee v$ and $C_2 \vee \neg v$, we call $C_1 \otimes_v C_2 = C_1 \vee C_2$ their *resolvent* with pivot v. The resolution rule states that $C_1 \vee v$ and $C_2 \vee \neg v$ imply their resolvent. Resolution is refutationally complete for propositional Boolean formulas, i.e. for every propositional Boolean formula that is equivalent to false we can derive the empty clause.

For *quantified* Boolean formulas, however, we need additional proof rules. The two most prominent ways to make resolution complete for QBF are to add either *universal reduction* or *universal expansion*, leading to the proof systems Q-resolution [28] and ∀Exp-Res [10, 29], respectively.

Universal expansion eliminates a single universal variable by creating two copies of the subformulas of its quantifier. Let $Q_1.\forall x.Q_2.\ \varphi$ be a QBF in PCNF, where Q_1 and Q_2 each are a sequence of quantifiers, and let Q_2 quantify over variables X. Universal expansion yields the *equivalent* formula $Q_1.Q_2.Q_2'.\ \varphi[\mathbf{1}/x, X'/X] \wedge \varphi[\mathbf{0}/x]$, where Q_2' is a copy of Q_2 but quantifying over a fresh set of variables X' instead of X. The term $\varphi[\mathbf{1}/x,\ X'/X]$ denotes the φ where x is replaced by $\mathbf{1}$ and the variables X are replaced by their counterparts in X'.

Universal reduction allows us to drop universal variables from clauses when none of the existential variables in that clause may depend on them. Let C a clause of a QBF and let l be a literal of a universally quantified variable in C. Let us further assume that $\bar{l}$ does not occur in C. If all existential variables v in C we have $\mathit{var}(l) \notin \mathit{dep}(v)$, universal reduction allows us to remove l from C. The resulting formula is equivalent to the original formula.

Stack. For convenience, we use a stack data structure to describe the algorithm. Formally, a stack is a finite sequence. Given a stack S, we use $S(i)$ to denote the i-th element of the stack, starting with index 0, and we use $S.S'$ to denote concatenation. We use $S[0, i]$ to denote the prefix up to element i of S. All stacks we consider are stacks of sets. In a slight abuse of notation, we also use stacks as the union of their elements when it is clear from the context. We also introduce an operation specific to stacks of sets S: We define $S.\mathit{add}(i, x)$ to be the stack that results from extending the set on level i by element x.

2.1 Unique Skolem Functions

Incremental determinization builds on the notion of unique Skolem functions. Let $\forall X.\exists Y.\ \varphi$ be a 2QBF in PCNF and let χ be a formula over X characterizing the *domain* of the Skolem functions we are currently interested in. We say that a variable $v \in Y$ has a *unique Skolem function* for domain χ, if for each assignment $\boldsymbol{x}$ with $\chi(\boldsymbol{x})$ there is a *unique* assignment $\boldsymbol{v}$ to v such that $\varphi(\boldsymbol{x}, \boldsymbol{v})$ is satisfiable. In particular, a unique Skolem function is a Skolem function:

Lemma 1. *If all existential variables have a unique Skolem function for the full domain $\chi = \mathbf{1}$, the formula is true.*

The semantic characterization of unique Skolem functions above does not help us with the computation of Skolem functions directly. We now introduce a local approximation of unique Skolem functions and show how it can be used as a propagation procedure.

We consider a set of variables $D \subseteq X \cup Y$ with $D \supseteq X$ and focus on the subset $\varphi|_D$ of clauses that only contain variables in D. We further assume that the existential variables in D already have unique Skolem functions for χ in the formula $\varphi|_D$. We now define how to extend D by an existential variable $v \notin D$. To define a Skolem function for v we only consider the clauses with *unique consequence* v, denoted $\mathcal{U}_v$, that contain a literal of v and otherwise only literals of variables in D. (Note that $\varphi|_D \cup \mathcal{U}_v = \varphi|_{D \cup \{v\}}$). We define that variable v has a *unique Skolem function relative to* D for χ, if for all assignments to D satisfying χ and φ there is a unique assignment to v satisfying $\mathcal{U}_v$.

In order to determine unique Skolem functions relative to a set D in practice, we split the definition into the two statements deterministic and unconflicted. Each statement can be checked by a SAT solver and together they imply that variable v has a unique Skolem function relative to D.

Given a clause C with unique consequence v, let us call $\neg(C \setminus \{v, \neg v\})$ the *antecedent* of C. Further, let $\mathcal{A}_l = \bigvee_{C \in \mathcal{U}_v, l \in C} \neg(C \setminus \{v, \neg v\})$ be the disjunction

of antecedents for the unique consequences containing the literal l of v. It is clear that whenever $\mathcal{A}_v$ is satisfied, v needs to be true, and whenever $\mathcal{A}_{\neg v}$ is satisfied, v need to be false. We define:

$$\begin{array}{lll} \mathsf{deterministic}(v,\varphi,\chi,D) := & \forall D.\ \varphi|_D \wedge \chi \ \Rightarrow & \mathcal{A}_v \vee \mathcal{A}_{\neg v} \\ \mathsf{unconflicted}(v,\varphi,\chi,D) := & \forall D.\ \varphi|_D \wedge \chi \ \Rightarrow & \neg(\ \mathcal{A}_v \wedge \mathcal{A}_{\neg v}\ ) \end{array}$$

deterministic states that v needs to be assigned either true or false for every assignment to D in the domain χ that is consistent with the existing Skolem function definitions $\varphi|_D$. Accordingly, unconflicted states that v does not have to be true and false at the same time (which would indicate a conflict) for any such assignment. Unique Skolem functions relative to a set D approximate unique Skolem functions as follows:

Lemma 2. *Let the existential variables in D have unique Skolem functions for domain χ and let $v \in Y$ have a unique Skolem function relative to D for domain χ. Then v has a unique Skolem function for domain χ.*

3 Inference Rules for Incremental Determinization

In this section, we develop a nondeterministic algorithm that formalizes and generalizes ID. We describe the algorithm in terms of inference rules that specify how the state of the algorithm can be developed. The state of the algorithm consists of the following elements:

- The solver status $S \in \{\mathsf{Ready}, \mathsf{Conflict}(L, \boldsymbol{x}), \mathsf{SAT}, \mathsf{UNSAT}\}$. The conflict status has two parameters: a clause L that is used to compute the learnt clause and the assignment $\boldsymbol{x}$ to the universals witnessing the conflict.
- A stack C of sets of clauses. $C(0)$ contains the original and the learnt clauses. $C(i)$ for $i > 0$ contain temporary clauses introduced by decisions.
- A stack D of sets of variables. The union of all levels in the stack represent the set of variables that currently have unique Skolem functions and the clauses in $C|_D$ represent these Skolem functions. $D(0)$ contains the universals and the existentials whose Skolem functions do not depend on decisions.
- A formula χ over $D(0)$ characterizing the set of assignments to the universals for which we still need to find a Skolem function.
- A formula α over variables $D(0)$ representing a *temporary* restriction on the domain of the Skolem functions.

We assume that we are given a 2QBF in PCNF $\forall X.\exists Y.\ \varphi$ and that all clauses in φ contain an existential variable. (If φ contains a non-tautological clause without existential variables, the formula is trivially false by universal reduction.) We define $(\mathsf{Ready}, \varphi, X, \mathbf{1}, \mathbf{1})$ to be the initial state of the algorithm. That is, the clause stack C initially has height 1 and contains the clauses of the formula φ. We initialize D as the stack of height 1 containing the universals.

Before we dive into the inference rules, we want to point out that some of the rules in this calculus may not be computable in polynomial time. The

$$\textsc{Propagate}\ \frac{(\mathsf{Ready}, C, D, \chi, \alpha) \qquad v \notin D \qquad \mathsf{deterministic}(v, C, \chi \wedge \alpha, D) \qquad \mathsf{unconflicted}(v, C, \chi \wedge \alpha, D)}{(\mathsf{Ready}, C, D.add(|D|-1, v), \chi, \alpha)}$$

$$\textsc{Decide}\ \frac{(\mathsf{Ready}, C, D, \chi, \alpha) \qquad v \notin D \qquad \text{all } c \in \delta \text{ have unique consequence } v}{(\mathsf{Ready}, C.\delta, D.\emptyset, \chi, \alpha)}$$

$$\textsc{Sat}\ \frac{(\mathsf{Ready}, C, D, \chi, \mathbf{1}) \qquad D = X \cup Y}{(\mathsf{SAT}, C, D, \chi, \mathbf{1})}$$

Fig. 1. Inference rules needed to prove true QBF

judgements deterministic and unconflicted require us to solve a SAT problem and are, in general, NP-complete. This is still easier than the 2QBF problem itself (unless NP includes Π_2^P) and in practice they can be discharged quickly by SAT solvers.

3.1 True QBF

We continue with describing the basic version of ID, consisting of the rules in Fig. 1 and Fig. 2, and first focus on the rules in Fig. 1, which suffice to prove true 2QBFs. Propagate allows us to add a variable to D, if it has a unique Skolem function relative to D. The judgements deterministic and unconflicted involve the current set of clauses C (i.e. the union of all sets of clauses in the sequence C). These checks are restricted to the domain $\chi \wedge \alpha$. Both χ and α are true throughout this section; we discuss their use in Section 4 and Section 5.

Invariant 1. All existential variables in D have a unique Skolem function for the domain $\chi \wedge \alpha$ in the formula $\forall X.\exists Y.\ C|_D$, where $C|_D$ are the clauses in C that contain only variables in D.

If Propagate identifies all variables to have unique Skolem functions relative to the growing set D, we know that they also have unique Skolem functions (Lemma 2). We can then apply Sat to reach the SAT state, representing that the formula has been proven true (Lemma 1).

Lemma 3. *ID cannot reach the SAT state for false QBF.*

Proof. Let us assume we reached the SAT state for a false 2QBF and prove the statement by way of contradiction. The SAT state can only be reached by the rule Sat and requires $D = X \cup Y$. By Invariant 1 all variables have a Skolem function in $\forall X.\exists Y.\ C$. Since C includes φ, this Skolem function does not violate any clause in φ, which means it is indeed a proof. □

$$\textsc{Conflict}\ \frac{(\mathsf{Ready}, C, D, \chi, \alpha) \qquad \boldsymbol{x} \text{ refutes } \mathsf{unconflicted}(v, C, \chi \wedge \alpha, D)}{(\mathsf{Conflict}(\{v, \neg v\}, \boldsymbol{x}), C, D, \chi, \alpha)}$$

$$\textsc{Analyze}\ \frac{(\mathsf{Conflict}(L, \boldsymbol{x}), C, D, \chi, \alpha) \qquad c \in C(0) \qquad l \in L \qquad \bar{l} \in c}{(\mathsf{Conflict}(L \otimes_{var(l)} c, \boldsymbol{x}), C, D, \chi, \alpha)}$$

$$\textsc{Learn}\ \frac{(\mathsf{Conflict}(L, \boldsymbol{x}), C, D, \chi, \alpha) \qquad var(L) \not\subseteq D}{(\mathsf{Ready}, C.add(0, L), D, \chi, \alpha)}$$

$$\textsc{Unsat}\ \frac{(\mathsf{Conflict}(L, \boldsymbol{x}), C, D, \chi, \alpha) \qquad var(L) \subseteq D(0) \qquad \boldsymbol{x} \not\models L}{(\mathsf{UNSAT}, C, D, \chi, \alpha)}$$

$$\textsc{Backtrack}\ \frac{(S, C, D, \chi, \alpha) \qquad 0 < dlvl \leq |C|}{(S, C[0, dlvl], D[0, dlvl], \chi, \alpha)}$$

Fig. 2. Additional inference rules needed to disprove false QBF

When Propagate is unable to determine the existence of a unique Skolem function (i.e. for variables where the judgement deterministic does not hold) we can use the rule Decide to introduce additional clauses such that deterministic holds and propagation can continue. Note that additional clauses make it easier to satisfy deterministic and adding the clause v (i.e. a unit clause) even *ensures* that deterministic holds for v.

Assuming we consider a true 2QBF, we can pick a Skolem function f_y for each existential variable y and encode it using Decide. We can simply consider the truth table of f_y in terms of the universal variables and define δ to be the set of clauses $\{\neg \boldsymbol{x} \vee v \mid f_y(\boldsymbol{x})\} \cup \{\neg \boldsymbol{x} \vee \neg v \mid \neg f_y(\boldsymbol{x})\}$. (Here we interpret the assignment $\boldsymbol{x}$ as a conjunction of literals.) These clauses have unique consequence v and they guarantee that v is deterministic. Further, they guarantee that v is unconflicted, as otherwise f_y would not be a Skolem function, so we can apply Propagate to add v to D. Repeating this process for every variable let us reach the point where $Y \subseteq D$ and we can apply Sat to reach the SAT state.

Lemma 4. *ID can reach the SAT state for true QBF.*

Note that proving the truth of a QBF in this way requires guessing correct Skolem functions for all existentials. In Subsection 3.4 we discuss how termination is guaranteed with a simpler type of decisions.

3.2 False QBF

To disprove false 2QBFs, i.e. formulas that do not have a Skolem function, we need the rules in Fig. 2 in addition to Propagate and Decide from Fig. 1.

The conflict state can only be reached via the rule Conflict, which requires that a variable v is conflicted, i.e. unconflicted fails. The Conflict rule stores the assignment $\boldsymbol{x}$ to D that proves the conflict and it creates the nucleus of the learnt clause $\{v, \neg v\}$. Via Analyze we can then resolve that nucleus with clauses in $C(0)$, which consists of the original clauses and the clauses learnt so far. We are allowed to add the learnt clause back to $C(0)$ by applying Learn.

Invariant 2. $C(0)$ is equivalent to φ.

Note that $C(0)$ and φ are propositional formulas over $X \cup Y$. Their equivalence means that they have the same set of satisfying assignments. We prove Invariant 2 together with the next invariant.

Invariant 3. Clause L in conflict state $\mathsf{Conflict}(L, \boldsymbol{x})$ is implied by φ.

Proof. $C(0)$ contains φ initially and is only ever changed by adding clauses through the Learn rule, so $C(0) \Rightarrow \varphi$ holds throughout the computation.

We prove the other direction of Invariant 2 and Invariant 3 by mutual induction. Initially, $C(0)$ consists exactly of the clauses φ, satisfying Invariant 2. The nucleus of the learnt clause $v \vee \neg v$ is trivially true, so it is implied by any formula, which gives us the base case of Invariant 3. Analyze is the only rule modifying L, and hence soundness of resolution together with Invariant 2 already gives us the induction step for Invariant 3 [30]. Since Learn is the only rule changing $C(0)$, Invariant 3 implies the induction step of Invariant 2. □

When adding the learnt clause to $C(0)$ we have to make sure that Invariant 1 is preserved. Learn hence requires that we have backtracked far enough with Backtrack, such that at least one of the variables in L is not in D anymore. In this way, L may become part of future Skolem function definitions, but will first have to be checked for causing conflicts by Propagate.

If all variables in L are in $D(0)$ and the assignment $\boldsymbol{x}$ from the conflict violates L, we can conclude the formula to be false using Unsat. The soundness of this step follows from the fact that $\boldsymbol{x}$ includes an assignment satisfying $C(0)|_{D(0)}$ (i.e. the clauses defining the Skolem functions for $D(0)$), Invariant 1 and Invariant 3.

Lemma 5. *ID cannot reach the UNSAT state for true QBF.*

We will now show that we can disprove any false QBF. The main difficulty in this proof is to show that from any Ready state we can learn a *new* clause, i.e. a clause that is semantically different to any clause in $C(0)$, and then return to the Ready state. Since there are only finitely many semantically different clauses over variables $X \cup Y$, and we cannot terminate in any other way (Lemma 5), we eventually have to find a clause L with $var(L) \subseteq D(0)$, which enables us to go to the UNSAT state.

From the Ready state, we can always add more variables to D with Decide and Propagate, until we reach a conflict. (Otherwise we would reach a state where $D = Y$ we were able to prove SAT, contradicting Lemma 5.) We only enter a Conflict state for a variable v, if there are two clauses $(c_1 \vee v)$ and $(c_2 \vee \neg v)$ with unique consequence v such that $\boldsymbol{x} \models \neg c_1 \wedge \neg c_2$ (see definition of unconflicted).

In order to apply ANALYZE, we need to make sure that $(c_1 \vee v)$ and $(c_2 \vee \neg v)$ are in $C(0)$. We can guarantee this by restricting DECIDE as follows: We say a decision for a variable v' is *consistent with the unique consequences* in state $(\mathsf{Ready}, C, D, \chi, \alpha)$, if $\mathsf{unconflicted}(v, C.\delta, \chi \wedge \alpha, D)$. We can construct such a decision easily by applying DECIDE only on variables that are not conflicted already (i.e. $\mathsf{unconflicted}(v, C, \chi \wedge \alpha, D)$) and by defining δ to be the CNF representation of $\neg \mathcal{A}_v \Rightarrow \neg v$ (i.e. require v to be false, unless a unique consequence containing literal v applies). It is clear that for this δ no new conflict for v is introduced and hence $\mathsf{unconflicted}(v, C.\delta, \chi \wedge \alpha, D)$.

Assuming that all decisions are taken consistent with the unique consequences, we know that when we encounter a conflict for variable v, we did not apply DECIDE for v, and hence the clauses $(c_1 \vee v)$ and $(c_2 \vee \neg v)$ causing the conflict must be in $C(0)$. We can hence apply ANALYZE twice with clauses $(c_1 \vee v)$ and $(c_2 \vee \neg v)$ and obtain the learnt clause $L = c_1 \vee c_2$. Since $\boldsymbol{x} \models \neg c_1 \wedge \neg c_2$, the learnt clause is violated by $\boldsymbol{x}$. As $\boldsymbol{x}$ refutes $\mathsf{unconflicted}(v, C, \chi \wedge \alpha, D)$ by construction, it must satisfy the clauses $C|_D$ and learnt clause L hence cannot be in $C|_D$. Further, L only contains variables that are in D, as $(c_1 \vee v)$ and $(c_2 \vee \neg v)$ were clauses with unique consequence v. So, L would have been in $C|_D$, if it existed in C already, and hence L is new. We can either add the new clause to $C(0)$ after backtracking, or we can conclude UNSAT.

Lemma 6. *ID can reach the* UNSAT *state for false QBF.*

The clause learning process considered here only applies one actual resolution step per conflict $(L_1 \otimes_v L_2)$. In practice, we probably want to apply multiple resolution steps before applying LEARN. It is possible to use the conflicting assignment $\boldsymbol{x}$ to (implicitly) construct an implication graph and mimic the clause learning of SAT solvers [8, 31].

3.3 Example

We now discuss the application of the inference rules along the following formula:

$$\begin{aligned}
\forall x_1, x_2.\ \exists y_1, \ldots, y_4.\ & (x_1 \vee \neg y_1) \ \wedge\ (x_2 \vee \neg y_1) \ \wedge\ (\neg x_1 \vee \neg x_2 \vee y_1) \wedge && (1)\\
& (\neg x_2 \vee y_2) \ \wedge\ (\neg y_1 \vee y_2) \ \wedge\ (x_2 \vee y_1 \vee \neg y_2) \wedge && (2)\\
& (y_1 \vee \neg y_3) \ \wedge\ (y_2 \vee \neg y_3) \ \wedge && (3)\\
& (\neg y_1 \vee y_4) \ \wedge\ (\neg y_3 \vee \neg y_4) && (4)
\end{aligned}$$

Initially, the state of the algorithm is the tuple $(\mathsf{Ready}, \varphi, X, \mathbf{1}, \mathbf{1})$. The rule PROPAGATE can be applied to y_1 in the initial state, as we are in the Ready state, $y_1 \notin X$, and because y_1 satisfies the checks $\mathsf{deterministic}$ and $\mathsf{unconflicted}$: The antecedents of y_1 are $\mathcal{A}_{y_1} = x_1 \wedge x_2$ and $\mathcal{A}_{\neg y_1} = \neg x_1 \vee \neg x_2$ (see clauses in line (1)). It is easy to check that both $\mathcal{A}_{y_1} \vee \mathcal{A}_{\neg y_1}$ nor $\neg(\mathcal{A}_{y_1} \wedge \mathcal{A}_{\neg y_1})$ hold for all assignments to x_1 and x_2. The state resulting from the application of PROPAGATE is $(\mathsf{Ready}, \varphi, X \cup \{y_1\}, \mathbf{1}, \mathbf{1})$. (Alternatively, we could apply DECIDE in the initial state, but deriving unique Skolem functions is generally preferable.)

While PROPAGATE was not applicable to y_2 before, it now is, as the increased set D made y_2 deterministic (see clauses in line (2)). We can thus derive the state $(\mathsf{Ready}, \varphi, X \cup \{y_1, y_2\}, \mathbf{1}, \mathbf{1})$.

Now, we ran out of variables to propagate and the only applicable rule is DECIDE. We arbitrarily choose y_3 as our decision variable and arbitrarily introduce a single clause $\delta = \{(\neg y_1 \vee \neg y_2 \vee y_3)\}$, arriving in the state $(\mathsf{Ready}, \varphi.\delta, X \cup \{y_1, y_2\}, \mathbf{1}, \mathbf{1})$. In this step we can immediately apply PROPAGATE (consider δ and the clauses in line (3)) to add the decision variable to the set D and arrive at $(\mathsf{Ready}, \varphi.\delta, X \cup \{y_1, y_2, y_3\}, \mathbf{1}, \mathbf{1})$.

We can now apply BACKTRACK to undo the last decision, but this would not be productive. Instead identify y_4 to be conflicted and we enter a conflict state with CONFLICT: $(\mathsf{Conflict}(\{y_4, \neg y_4\}, x_1 \wedge x_2), \varphi.\delta, X \cup \{y_1, y_2, y_3\}, \mathbf{1}, \mathbf{1})$. To resolve the conflict we apply ANALYZE twice - once with each of the clauses in line (4) - bringing us into state $(\mathsf{Conflict}(\{\neg y_1, \neg y_3\}, x_1 \wedge x_2), \varphi.\delta, X \cup \{y_1, y_2, y_3\}, \mathbf{1}, \mathbf{1})$. We can backtrack one level such that $D = X \cup \{y_1, y_2\}$ and then apply LEARN to enter state $(\mathsf{Ready}, \varphi \cup \{(\neg y_1 \vee \neg y_3)\}, X \cup \{y_1, y_2\}, \mathbf{1}, \mathbf{1})$.

The rest is simple: we apply PROPAGATE on y_3 and take a decision for y_4. As no other variable can depend on y_4 we can take an arbitrary decision for y_4 that makes y_4 deterministic, as long as this does not make y_4 conflicted. Finally, we can propagate y_4 and then apply SAT to conclude that we have found Skolem functions for all existential variables.

3.4 Termination

So far, we have described sound and nondeterministic algorithms that allow us to prove or disprove any 2QBF. We can easily turn the algorithm in the proof of Lemma 6 into a *deterministic* algorithm that terminates for both true and false QBF by introducing an arbitrary ordering of variables and assignments: Whenever there is nondeterminism in the application of one of the rules as described in Lemma 6, pick the smallest variable for which one of the rules is applicable. When multiple rules are applicable for that variable, pick them in the order they appear in the figures. When the inference rule allows multiple assignments, pick the smallest. In particular, this guarantees that the existential variables are added to D in the arbitrarily picked order, as for any existential not in D we can either apply PROPAGATE, DECIDE, or CONFLICT.

Restricting DECIDE to decisions that are consistent with the unique consequences may be unintuitive for true QBF, where we try to find a Skolem function. However, whenever we make the 2QBF false by introducing clauses with DECIDE, we will eventually go to a conflict state and learn a new clause. Deriving the learnt clause for conflicted variable v from two clauses with unique consequence v (as described for Lemma 6) means that we push the constraints towards *smaller* variables in the variable ordering. The learnt clause will thus improve the Skolem function for a smaller variable or cause another conflict for a smaller variable. In the extreme case, we will eventually learn clauses that look like function table entries, as used in Lemma 4, i.e. clauses containing exactly one existential variable. At some point, even with our restriction for DECIDE, we

	SAT $\exists Y.\ \varphi$	2QBF $\forall X.\, \exists Y.\ \varphi$
State	Partial assignment of values	Partial assignment of functions
Propagation	unit propagation	unique Skolem function w.r.t. D
Decision	unit clause	clause with unique consequence
Conflict	unit clauses y and $\neg y$	$\exists X$ that implies y and $\neg y$
Learning	clause	clause

Fig. 3. Concepts in ID and their counterparts in CDCL

cannot make a "wrong" decision: The cases for which a variable does not have a clause with unique consequence are either irrelevant for the satisfaction of the 2QBF or our restricted decisions happen to make the right assignment.

In cases where no static ordering of variables is used - as it will be the case in any practical approach - the termination for true QBF is less obvious but follows the same argument: Given enough learnt clauses, the relationships between the variables are dense enough such that even naive decisions suffice.

3.5 Pure literals

The original paper on ID introduces the notion of *pure literals* for QBF that allows us to propagate a variable v even if it is not deterministic, if for a literal l of v, all clauses c that l occurs in are either satisfied or l is the unique consequence of c. The formalization presented in this section allows us to conclude that pure literals are a special case of DECIDE: We can introduce clauses defining v to be of polarity $\bar{l}$ whenever all clauses containing l are satisfied by another literal.

That is, we can precisely characterize the minimal set of cases in which v has to be of polarity l and the decision is guaranteed to never introduce unnecessary conflicts. The same definition cannot be made when l occurs in clauses where it is not a unique consequence, as then the clause contains another variable that is not deterministic yet.

3.6 Relation of ID and CDCL

There are some obvious similarities between ID and conflict-driven clause learning (CDCL) for SAT. Both algorithms modify their partial assignments by propagation, decisions, clause learning, and backtracking. The main difference between the algorithms is that, while CDCL solvers maintain a partial assignment of Boolean values to variables, ID maintains a partial assignment of functions to variables (which is represented by the clauses $C|_D$). We summarized our observations in Fig. 3.

4 Inductive Reasoning

The CEGIS approach to solving a 2QBF $\forall X.\, \exists Y.\, \varphi$ is to iterate over X assignments $\boldsymbol{x}$ and check if there is an assignment $\boldsymbol{y}$ such that $\varphi(\boldsymbol{x}, \boldsymbol{y})$ is valid.

$$\textsc{InductiveRefinement}\ \frac{(\mathsf{Conflict}(L,\boldsymbol{x}),C,D,\chi,\alpha) \qquad \boldsymbol{y} \models \varphi(\boldsymbol{x}|_X)}{(\mathsf{Conflict}(L,\boldsymbol{x}) \text{ or } \mathsf{Ready},C,D,\chi \wedge \neg\varphi(X,\boldsymbol{y}),\alpha)}$$

$$\textsc{Failed}\ \frac{(\mathsf{Conflict}(L,\boldsymbol{x}),C,D,\chi,\alpha) \qquad \varphi(\boldsymbol{x}|_X) \text{ is unsatisfiable}}{(\mathrm{UNSAT},C,D,\chi,\alpha)}$$

Fig. 4. Inference rules adding inductive reasoning to ID

Upon every successful iteration we exclude all assignments to X for which $\boldsymbol{y}$ is a matching assignment. If the space of X assignments is exhausted we conclude the formula is true, and if we find an assignment to X for which there is no matching Y assignment, the formula is false [21].

While this approach shows poor performance on some problems, as discussed in the introduction, it is widely popular and has been successfully applied in many cases. In this section we present a way how it can be integrated in ID in an elegant way. The simplicity of the CEGIS approach carries over to our extension of ID - we only need the two additional inference rules in Fig. 4.

We exploit the fact that ID already generates assignments $\boldsymbol{x}$ to X in its conflict check. Whenever ID is in a conflict state, the rules in Fig. 4 allow us to check if there is an assignment $\boldsymbol{y}$ to Y matching $\boldsymbol{x}$. If there is such an assignment $\boldsymbol{y}$, we can use the soundness argument of CEGIS and exclude $\boldsymbol{x}$ and any other assignment to X to which $\boldsymbol{y}$ is a solution. The component χ of the solver state keeps track of domain (i.e. assignments to X) for which we still need to find a Skolem function. InductiveRefinement removes $\varphi(X,\boldsymbol{y})$ from χ, representing that the constant function $\boldsymbol{y}$ is a Skolem function for the domain $\varphi(X,\boldsymbol{y})$.

This gives rise to a new invariant, stating that $\neg\chi$ only includes assignments to X for which we know that there is a Y satisfying φ. With this invariant it is clear that Lemma 3 also holds for arbitrary χ.

Invariant 4. $\forall X.\exists Y.\ \neg\chi \Rightarrow \varphi$

Upon an InductiveRefinement step, we can choose to either continue in the conflict state (and learn a clause), or in the Ready state. Either way, the assignment $\boldsymbol{x}$ that provoked the previous conflicts is now excluded from the domain and cannot provoke any more conflicts. Thereby, we may even be able to propagate the variable for which we detected a conflict earlier.

It is easy to check that Propagate preserves Invariant 1 also if χ and α are not $\mathbf{1}$. Invariant 2 and Invariant 3 are unaffected by the rules in this section. To make sure that Lemma 5 is preserved as well, we thus only have to inspect Failed, which is trivially sound.

A portfolio approach? In principle, we could generate assignments $\boldsymbol{x}$ independently from the conflict check of ID. The result would be a portfolio approach that simply executes ID and CEGIS in parallel and takes the result from whichever method terminates first. The idea behind our extension is that conflict assign-

$$\textsc{Assume}\ \frac{(\mathsf{Ready}, C, D, \chi, \alpha) \qquad var(l) \in D(0)}{(\mathsf{Ready}, C, D, \chi, \alpha \wedge l)}$$

$$\textsc{Close}\ \frac{(\mathsf{Ready}, C, D, \chi, \alpha) \qquad D = X \cup Y}{(\mathsf{Ready}, C(0), D(0), \chi \wedge \neg\alpha, \mathbf{1})}$$

Fig. 5. Inference rules adding case distinctions to ID

ments are more selective and may thus increase the probability that we hit a refuting assignment to X. Also ID may profit from excluding groups of assignments for which frequently cause conflicts. We revisit this question in Section 6.

Example. We extend the example from Subsection 3.3 from the point where we entered the conflict state $(\mathsf{Conflict}(\{y_4, \neg y_4\}, x_1 \wedge x_2), \varphi.\delta, X \cup \{y_1, y_2, y_3\}, \mathbf{1}, \mathbf{1})$. We can apply InductiveRefinement, checking that there is indeed a solution to φ for the assignment x_1, x_2 to the universals (e.g. $y_1, y_2, \neg y_3, y_4$). We have the choice to either additionally do standard conflict analysis and learn a clause, or to go back to a Ready state - and effectively ignore the conflict. Let us try the latter, and go to state $(\mathsf{Ready}, \varphi.\delta, X \cup \{y_1, y_2, y_3\}, \neg x_1 \vee \neg x_2, \mathbf{1})$. Now, deterministic and unconflicted can never show the assignment x_1, x_2 again and, in fact, y_4 will never be conflicted again.

5 Expansion

Universal expansion (defined in Section 2) is another fundamental proof rule that deals with universal variables. It has been used in early QBF solvers [10] and has later been integrated in CEGAR-style QBF solvers [26, 32].

One way to look at the expansion of a universal variable x is that it introduces a case distinction over the possible values of x in the Skolem functions. However, instead of creating a copy of the formula explicitly, which often caused a blowup in required memory, we can reason about the two cases sequentially. The rules in Fig. 5 extend ID by universal expansion in this spirit.

Using Assume we can, at any point, assume that a variable v in $D(0)$, i.e. a variable that has a unique Skolem function without any decisions, has a particular value. This is represented by extending α by the corresponding literal of v, which restricts the domain of the Skolem function that we try to construct for subsequent deterministic and unconflicted checks. Invariant 1 and Lemma 5 already accommodate the case that α is not $\mathbf{1}$.

When we reach a point where D contains all variables, we cannot apply Sat, as that requires α to be true. In this case, Invariant 1 only guarantees us that the function we constructed is correct on the domain $\chi \wedge \alpha$. We can hence restrict the domain for which we still need to find a Skolem function and strengthen χ by $\neg\alpha$. In particular, Close maintains Invariant 4. When χ ends up being

equivalent to $\mathbf{0}$, Invariant 4 guarantees that the original formula is true. (In this case we can reach the SAT state easily, as we know that from now on every application of PROPAGATE must succeed. [1])

Note that ASSUME does not restrict us to assumptions on single variables. Together with DECIDE and PROPAGATE it is possible to introduce variables with arbitrary definitions, add them to $D(0)$, and then assume an outcome with the rule ASSUME.

Example. Again, we consider the formula from Subsection 3.3. Instead of the reasoning steps described in Subsection 3.3, we start using ASSUME with literal x_2. Whenever checking deterministic or unconflicted in the following, we will thus restrict ourselves to universal assignments that set x_2 to true. It is easy to check that this allows us to propagate not only y_1 and y_2, but also y_3. A decision (e.g. $\delta' = \{(y_4)\}$) for y_4 allows us to also propagate y_4 (this time without potential for conflicts), arriving in state $(\mathsf{Ready}, \varphi.\delta', X \cup \{y_1, y_2, y_3, y_4\}, \mathbf{1}, x_2)$.

We can CLOSE this case concluding that under the assumption x_2 we have found a Skolem function. We enter the state $(\mathsf{Ready}, \varphi, X, \neg x_2, \mathbf{1})$ which indicates that in the future we only have to consider universal assignments with $\neg x_2$. Also for the case $\neg x_2$, we cannot encounter conflicts for this formula. Expansion hence allows us to prove this formula without any conflicts.

6 Experimental Evaluation

We extended the QBF solver CADET [8] by the extensions described in Section 4 and Section 5. We use the CADET-IR and CADET-E to denote the extensions of CADET by inductive reasoning (Section 4) and universal expansion (Section 5), respectively. We also combined both extensions and refer to this version as CADET-IR-E. The experiments in this section evaluate these extensions against the basic version of CADET and against other successful QBF solvers of the recent years, in particular GhostQ [33], RAReQS [32], Qesto [23], DepQBF [19] in version 6, and CAQE [24, 26]. For every solver except CADET and GhostQ, we use Bloqqer [34] in version 031 as preprocessor. For our experiments, we used a machine with a 3.6 GHz quad-core Intel Xeon processor and 32 GB of memory. The timeout and memout were set to 600 seconds and 8 GB. We evaluated the solvers on the benchmark sets of the last competitive evaluation of QBF solvers, QBFEval-2017 [9].

How does inductive reasoning affect the performance? In Fig. 6 we see that CADET-IR clearly dominates plain CADET. It also dominates all solvers that relied on clause-level CEGAR and Bloqqer (CAQE, Qesto, RAReQS).

Only GhostQ beats CADET-IR and solves 5 more formulas (of 384). A closer look at the experimental data revealed that there are many formulas for which

[1] Technically, we could replace SAT by a rule that allows us to enter the SAT state whenever χ is $\mathbf{0}$, which arguably would be more elegant. But that would require us to introduce the CLOSE rule already for the basic ID inference system.

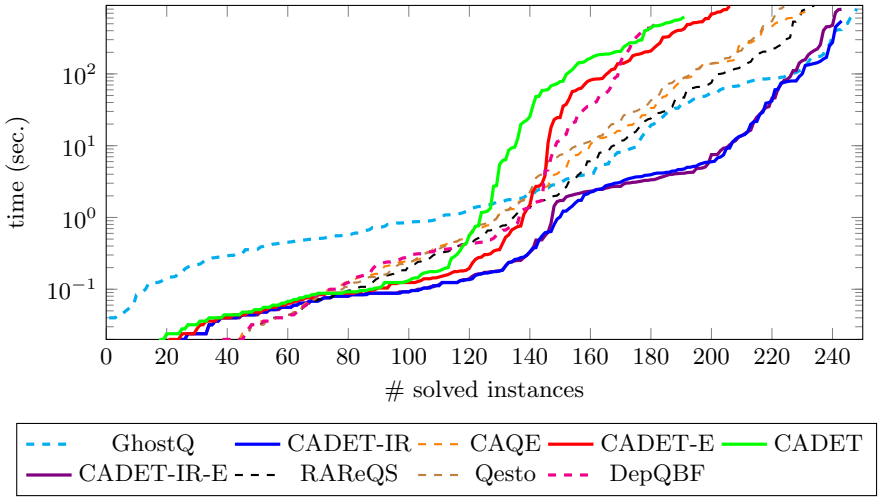

Fig. 6. Cactus plot comparing solvers on the QBFEval-2017 2QBF benchmark.

CADET-IR and GhostQ show widely different runtimes hinting at potential for future improvement.

GhostQ is based on the CEGAR principle, but reconstructs a circuit representation from the clauses instead of operating on the clauses directly [33]. This makes GhostQ a representative of QBF solvers working with so called "structured" formulas (i.e. not CNF). CADET, on the other hand, refrains from identifying logic gates in CNF formulas and directly operates with the "unstructured" CNF representation. In the ongoing debate in the QBF community on the best representation of formulas for solving quantified formulas, our experimental findings can thus be interpreted as a tie between the two philosophies.

Is the inductive reasoning extension just a portfolio-approach? To settle this question, we created a version of CADET-IR, called IR-only, that exclusively applies inductive reasoning by generating assignments to the universals and applying InductiveReasoning. This version of CADET does not learn any clauses, but otherwise uses the same code as CADET-IR. On the QBFEval-2017 benchmark, IR-only and CADET together solved 235 problems within the time limit, while CADET-IR solved 243 problems. That is, even though the combined runtime of CADET and IR-only was twice the runtime of CADET-IR, they solved fewer problems. CADET-IR also uniquely solved 22 problems. This indicates that CADET-IR improves over the portfolio approach.

How does universal expansion affect the performance? CADET-E clearly dominates plain CADET on QBFEval-2017, but compared to CADET-IR and some of the other QBF solvers, CADET-E shows mediocre performance overall. However, for some subsets of formulas, such as the Hardware Fixpoint formulas shown in Fig. 7, CADET-E dominated CADET, CADET-IR, and all other solvers. We also combined the two extensions of CADET to obtain CADET-IR-E. While this

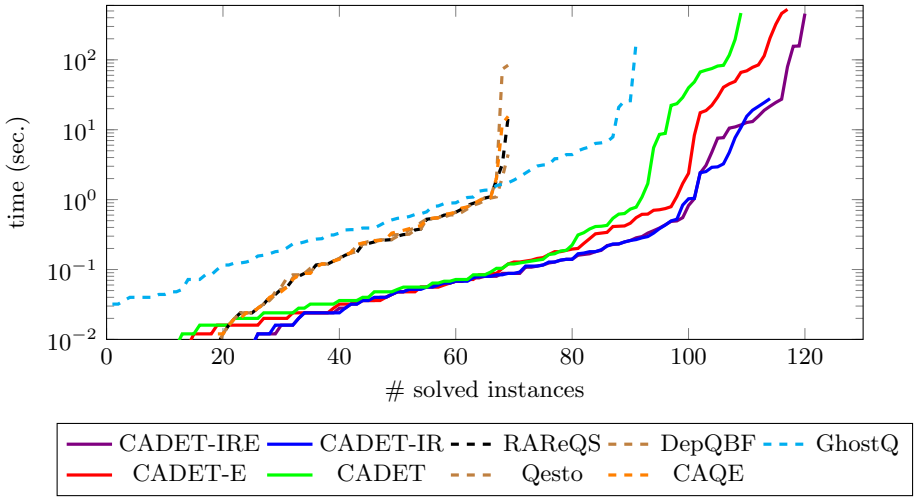

Fig. 7. Cactus plot comparing solver performance on the Hardware Fixpoint formulas. Some but not all of these formulas are part of QBFEval-2017. The formulas encode diameter problems that are known to be hard for classical QBF search algorithms [35].

helped to improve the performance on the Hardware Fixpoint formulas even further, it did not change the overall picture on QBFEval-2017.

7 Conclusion

Reasoning in quantified logics is one of the major challenges in computer-aided verification. Incremental Determinization (ID) introduced a new algorithmic principle for reasoning in 2QBF and delivered first promising results [8]. In this work, we formalized and generalized ID to improve the understanding of the algorithm and to enable future research on the topic. The presentation of the algorithm as a set of inference rules has allowed us to disentangle the design choices from the principles of the algorithm (Section 3). Additionally, we have explored two extensions of ID that both significantly improve the performance: The first one integrates the popular CEGAR-style algorithms and Incremental Determinization (Section 4). The second extension integrates a different type of reasoning termed universal expansion (Section 5).

Acknowledgements We want to thank Martina Seidl, who brought up the idea to formalize ID as inference rules, and Vijay D'Silva, who helped with disentangling the different perspectives on the algorithm. This work was supported in part by NSF grants 1139138, 1528108, 1739816, SRC contract 2638.001, the Intel ADEPT center, and the European Research Council (ERC) Grant OSARES (No. 683300).

References

1. A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu, "Programming by sketching for bit-streaming programs," in *Proceedings of PLDI*, 2005, pp. 281–294.
2. R. Alur, R. Bodik, G. Juniwal, M. M. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa, "Syntax-guided synthesis," *Dependable Software Systems Engineering*, vol. 40, pp. 1–25, 2015.
3. P. Faymonville, B. Finkbeiner, M. N. Rabe, and L. Tentrup, "Encodings of bounded synthesis," in *Proceedings of TACAS*, 2017.
4. R. Bloem, R. Könighofer, and M. Seidl, "SAT-based synthesis methods for safety specs," in *Proceedings of VMCAI*, K. L. McMillan and X. Rival, Eds. Springer Berlin Heidelberg, 2014, pp. 1–20.
5. A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat, "Combinatorial sketching for finite programs," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM Press, October 2006, pp. 404–415.
6. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proceedings of Computer Aided Verification*. Springer Berlin Heidelberg, 2000, pp. 154–169.
7. S. Jha and S. A. Seshia, "A Theory of Formal Synthesis via Inductive Learning," *Acta Informatica*, vol. 54, no. 7, pp. 693–726, 2017.
8. M. N. Rabe and S. A. Seshia, "Incremental determinization," in *Proceedings of SAT*. Berlin, Heidelberg: Springer-Verlag, 2016.
9. L. Pulina, "The ninth QBF solvers evaluation - preliminary report," in *Proceedings of QBF@SAT*, ser. CEUR Workshop Proceedings, vol. 1719. CEUR-WS.org, 2016, pp. 1–13.
10. A. Biere, "Resolve and expand," in *Proceedings of SAT*, 2004.
11. F. Pigorsch and C. Scholl, "An AIG-based QBF-solver using SAT for preprocessing," in *Proceedings of DAC*. IEEE, 2010, pp. 170–175.
12. G. Charwat and S. Woltran, "Dynamic programming-based QBF solving," in *Proceedings of Quantified Boolean Formulas*, ser. CEUR Workshop Proceedings, F. Lonsing and M. Seidl, Eds., vol. 1719, 2016, pp. 27–40.
13. P. Baumgartner and C. Tinelli, "The model evolution calculus," in *Proceedings of CADE*, F. Baader, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 350–364.
14. R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Abstract DPLL and abstract DPLL modulo theories," in *Proceedings of LPAR*, F. Baader and A. Voronkov, Eds. Springer Berlin Heidelberg, 2005, pp. 36–50.
15. L. de Moura and D. Jovanović, "A model-constructing satisfiability calculus," in *Verification, Model Checking, and Abstract Interpretation*, R. Giacobazzi, J. Berdine, and I. Mastroeni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–12.
16. K. Fazekas, M. Seidl, and A. Biere, "A duality-aware calculus for quantified Boolean formulas," in *Proceedings of SYNASC*. IEEE Computer Society, 2016, pp. 181–186.
17. V. D'Silva, L. Haller, and D. Kroening, "Abstract conflict driven learning," in *Proceedings POPL*. ACM, 2013, pp. 143–154.
18. E. Giunchiglia, M. Narizzano, and A. Tacchella, "QuBE: A system for deciding quantified Boolean formulas satisfiability." in *Proceedings of IJCAR*, 2001, pp. 364–369.

19. F. Lonsing and A. Biere, "DepQBF: A dependency-aware QBF solver," *JSAT*, vol. 7, no. 2-3, pp. 71–76, 2010.
20. D. Ranjan, D. Tang, and S. Malik, "A comparative study of 2QBF algorithms," in *Proc. of SAT*. ACM, 2004.
21. M. Janota and J. P. M. Silva, "Abstraction-based algorithm for 2QBF," in *Proceedings of SAT*, 2011, pp. 230–244.
22. M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, "Solving QBF with counterexample guided refinement." in *Proceedings of SAT*, 2012, pp. 114–128.
23. M. Janota and J. Marques-Silva, "Solving QBF by clause selection," in *Proceedings of IJCAI*. AAAI Press, 2015, pp. 325–331.
24. M. N. Rabe and L. Tentrup, "CAQE: A certifying QBF solver," in *Proceedings of FMCAD*, 2015, pp. 136–143.
25. R. Bloem, N. Braud-Santoni, and V. Hadzic, "QBF solving by counterexample-guided expansion," *CoRR*, vol. abs/1611.01553, 2016. [Online]. Available: http://arxiv.org/abs/1611.01553
26. L. Tentrup, "On expansion and resolution in CEGAR based QBF solving," in *Proceedings of CAV*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 475–494.
27. G. S. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in constructive mathematics and mathematical logic, Reprinted in [36]*, vol. 2, no. 115-125, pp. 10–13, 1968.
28. H. Buning, M. Karpinski, and A. Flogel, "Resolution for quantified Boolean formulas," *Information and Computation*, vol. 117, no. 1, pp. 12 – 18, 1995.
29. M. Janota and J. Marques-Silva, "Expansion-based QBF solving versus Q-resolution," *Theoretical Computer Science*, vol. 577, no. 0, pp. 25–42, April 2015.
30. J. A. Robinson, "A machine-oriented logic based on the resolution principle," *J. ACM*, vol. 12, no. 1, pp. 23–41, Jan. 1965.
31. J. P. Marques-Silva and K. A. Sakallah, "GRASP - A new search algorithm for satisfiability," in *Proceedings of CAD*. IEEE, 1997, pp. 220–227.
32. M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, "Solving QBF with counterexample guided refinement," *Artif. Intell.*, vol. 234, pp. 1–25, 2016.
33. W. Klieber, S. Sapra, S. Gao, and E. Clarke, "A non-prenex, non-clausal QBF solver with game-state learning," in *Proceedings of SAT*. Springer-Verlag, 2010, pp. 128–142.
34. A. Biere, F. Lonsing, and M. Seidl, "Blocked clause elimination for QBF," in *Proceedings of CADE*, 2011, pp. 101–115.
35. D. Tang, Y. Yu, D. Ranjan, and S. Malik, "Analysis of search based algorithms for satisfiability of propositional and quantified Boolean formulas arising from circuit state space diameter problems," in *Proceedings of SAT*, H. H. Hoos and D. G. Mitchell, Eds., 2005, pp. 292–305.
36. J. Siekmann and G. Wrightson, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Springer Science & Business Media, 1983.
37. M. N. Rabe, "CADET," https://github.com/MarkusRabe/cadet, 2018.
38. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings DAC*. ACM, 2001, pp. 530–535.
39. A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009.

A How does CADET implement the inference rules?

Any implementation of ID has to resolve the nondeterministic choices offered by the inference rules presented in the previous sections. In this section, we first discuss how CADET resolved the major design choices for its implementation of ID [37]. In the second part of this section we then discuss how we implemented the extensions discussed in Section 4 and Section 5.

Propagation first. The first major design choice of CADET is to propagate whenever possible. CADET starts solving formulas with checking all variables for the potential to assign them a unique Skolem function through PROPAGATE and does so after every decision. To keep computational cost low, CADET only checks deterministic for variables that do occur as the unique consequence of at least one clause. Further, the check deterministic is approximated by dropping $\varphi|_D$ to reduce the cost of the check. The check for conflicts is applied cautiously: unconflicted is only tested for variables that pass the check deterministic. This way, CADET may find some conflicts much later but saves many conflict checks.

Decision variables. DECISION provides a huge degree of freedom. We have to select a decision variable v and additionally we have to select a set of constraints δ for v. CADET mimics SAT solvers and uses a VSIDS heuristic, which was pioneered by the solver Chaff and is still popular in the SAT community [38]. That is, for every variable CADET maintains an activity value that gets increased by 1 whenever the variable is resolved in the ANALYZE rule. After every conflict, the activity values of all variables get decreased by a small factor.

In contrast to SAT solvers, ID offers more than just two possible "values" for the decision variable v. Any of the $2^{2^{|X|}}$ Skolem functions for v may be a valid choice at this point and selecting the "correct" function could be as hard as solving the formula itself. *Instead of constructing a "good" function, CADET chooses to construct the most simple function that still guarantees progress.* Given a variable to take a decision for, CADET chooses a Boolean value b (true by default) and defines v to be b for all cases in which none of the clauses with unique consequence applies. Progress is guaranteed, as the clauses learnt for decisions of this type are guaranteed to be new. The finite number of semantically different clauses means that the method terminates eventually.

Clause learning. CADET again borrows from SAT solvers and analyzes conflicts along an implication graph as introduced in the seminal CDCL paper [31]. CADET stores the clauses with unique consequence for every successful PROPAGATE step and only uses these unique consequences as pivot elements for the resolution in ANALYZE. Additionally, CADET restricts the resolution to clauses whose antecedent is satisfied by the conflicting assignment and whose unique consequence is on the same decision level as the conflict.

Restarts. Upon a conflict, it makes sense to backtrack the smallest number of decision levels in order to maintain more of the Skolem functions that we have derived already. However, in certain intervals, CADET chooses to backtrack to decision level 0 instead, which resembles *restarts* in SAT solvers.

Discussion and open questions. While it is understandable that the first implementation of ID sticks close to the design choices of SAT solvers, we believe that there is potential for future improvement. After all, the relative runtimes of the different inference steps of ID might substantially differ from those of CDCL. For example, the propagation step in ID is an NP-hard operation - far costlier than unit propagation. It may pay off to identify cheaper approximations that reduce the time spent in propagation.

A.1 Implementation of inductive reasoning

The simplicity of CEGAR-based QBF solvers carries over to the integration of inductive reasoning in ID. Upon every conflict, we apply InductiveRefinement once and continue deriving a conflict clause as normal. For that we need to maintain a SAT solver instance holding the clauses of the original 2QBF formula. Using the incremental interface of modern SAT solvers, we assume the assignment to the universal variables and ask the solver for an assignment to the remaining (existential) variables. If the SAT solver returns such an assignment, we apply standard generalization techniques to remove universal variables from the refinement that do not contribute to satisfaction of any clause.

A.2 Implementation of universal expansion

The major question when implementing Assume is when to introduce assumptions and which assumptions to select. Introducing too many assumptions can be quite costly, as we then end up constructing a Skolem function that is only valid for a small portion of its domain. To limit the number of assumptions by selecting exactly one literal after every restart.

After experimenting with various heuristics, we settled for a lookahead-style variable selection [39, Chapter 5]. We believe that the potential cost of unnecessary assumptions justifies the computational cost of this heuristic. We determine the number of propagations of constants and propagations of Skolem functions following the assignment of the variable. To score a variable, we determine the propagation counts for both polarities of the variable, c_1 and c_0, and compute $(c_1c_2+c_1+c_2+1)(a+1)$, where a is the variable's activity. That is, the product of propagation counts typically is the dominating factor, such that we mostly select variables that have a widespread effect on the problem when assigned either way. This approach has the additional advantage that we sometimes encounter inconsistent assumptions, meaning that we get can add some assumptions without the risk of wasting effort.