A CEGAR Approach to QBF

Solving Quantified Boolean Formulas (QBF) with Counterexample-Guided Abstraction Refinement (CEGAR)

Mikoláš Janota, Will Klieber*, Joao Marques-Silva, Edmund M. Clarke

*Presenting
Outline

- Introduction to QBF
- CEGAR for 2QBF
- CEGAR for general QBF and optimizations
- Incorporating CEGAR in DPLL solvers
- Experimental results
Motivation

- **Why study Quantified Boolean Formulas (QBF)?**
  - Practical problems naturally expressed in QBF.
  - Formal Verification: Bounded Model Checking (BMC).
  - Advances in QBF enable faster verification of larger systems.

- **Difficulty of QBF:**
  - PSPACE-complete.
  - Worst case likely exponential in number of variables.
  - For verification: Must give provably correct answer.
  - Industrial verification instances often easier than worst case.
What are quantified boolean formulas (QBF)?

- Extension of propositional logic.
  - Boolean variables can be quantified.

- “∃x. Φ” means “there exists a value of x such that Φ is true”.
  - Example 1: ∃x.∃y. x ∧ y is true, because x ∧ y is true if x=True and y=True.

- “∀x. Φ” means “for all values of x, Φ is true”.
  - Example 2: ∀x.∀y. x ∨ y is false, because x ∨ y is false if x=False and y=False.

- We’ll use the symbol “Φ” to denote a QBF formula.
What are quantified boolean formulas (QBF)?

- BNF formal grammar:

\[ \Phi ::= \text{True} \mid \text{False} \mid x \mid \neg x \mid \Phi \land \Phi \mid \Phi \lor \Phi \mid \exists x. \Phi \mid \forall x. \Phi \]

- A QBF formula can be:
  - The constant True or the constant False.
  - A boolean variable, \( x \).
  - The negation of a boolean variable, \( \neg x \).
  - A conjunction \( \Phi_1 \land \Phi_2 \) or disjunction \( \Phi_1 \lor \Phi_2 \).
  - An existentially quantified formula \( \exists x. \Phi \) or a universally quantified formula \( \forall x. \Phi \).
  - Variable shadowing, e.g., \( \exists x. (\exists x. x) \), is disallowed.
What are quantified boolean formulas (QBF)?

- **Semantics:**
  - $\forall x. \Phi = (\Phi|_{x:True}) \land (\Phi|_{x:False})$
  - $\exists x. \Phi = (\Phi|_{x:True}) \lor (\Phi|_{x:False})$
  - where $\Phi|_{x:C}$ denotes substitution of $x$ with $C$ in $\Phi$. 

- Closed QBF: Every variable is quantified, and no variable occurs outside the scope of its quantifier. We only consider closed QBF.

- Example A: $\forall y. x \land \exists x. y$ is not closed.

- Example B: $\forall y. y \land \exists x. x$ is closed.

- A closed QBF evaluates to either True or False.

- Goal of QBF solver: determine this truth value.
What are quantified boolean formulas (QBF)?

Semantics:

\[ \forall x. \Phi = (\Phi|_{\{x: \text{True}\}}) \land (\Phi|_{\{x: \text{False}\}}) \]
\[ \exists x. \Phi = (\Phi|_{\{x: \text{True}\}}) \lor (\Phi|_{\{x: \text{False}\}}) \]

where \( \Phi|_{\{x: C\}} \) denotes substitution of \( x \) with \( C \) in \( \Phi \).

Closed QBF: Every variable is quantified, and no variable occurs outside the scope of its quantifier. We only consider closed QBF.

- Example A: \( \forall y. x \land \exists x. y \) is not closed.
- Example B: \( \forall y. y \land \exists x. x \) is closed.

A closed QBF evaluates to either True or False.

Goal of QBF solver: determine this truth value.
What are quantified boolean formulas (QBF)?

- **Quantifier Blocks.** We may group quantifiers into blocks of the same type (existential or universal).
  - Let $X = \{x_1, \ldots, x_n\}$
  - $QX.\Phi$ means $Q x_1 \ldots Q x_n. \Phi$, where $Q$ is either $\forall$ or $\exists$.
    - $\exists X.\Phi$ means $\exists x_1 \ldots \exists x_n. \Phi$.
    - $\forall X.\Phi$ means $\forall x_1 \ldots \forall x_n. \Phi$.

- **Prenex Form:** $Q_1 X_1 \ldots Q_n X_n. \phi$ where $\phi$ has no quantifiers.
  - Input is in prenex format (QDIMACS).
  - Non-prenex formulas arise in CEGAR process.
QBF as a Game

- Existentially quantified variables are owned by Player $\exists$. Universally quantified variables are owned by Player $\forall$.
- Players assign variables in quantification order.
  - Start with outermost quantified (leftmost).
- We say Player $\exists$ wins a closed formula $\Phi$ iff $\Phi$ is true. We say Player $\forall$ wins a closed formula $\Phi$ iff $\Phi$ is false.
Approaches to QBF

- DPLL
  - E.g., Qube

- Expansion-based
  - E.g., Quantor and Nenofex
  - Shannon expansion of variable from inner quantifier block

- Skolemizing
  - E.g., Skizzo

- Portfolio
  - E.g., AQME

- Counterexample-Guided Abstraction Refinement (CEGAR)
Consider $\exists X. \forall Y. \phi$ where $\phi$ has no quantifiers.

We say a move for $Y$ is an assignment to $Y$. (Game view)

Notation: Let $\text{moves}(Y)$ denote the set of all assignments to $Y$.

Given a move $\mu$, we write “$\phi|\mu$” to denote substitution under $\mu$.

Example:

Let $\phi = (x \land y) \lor (\neg x \land \neg y)$.

If $\mu = \{y : \text{False}\}$, then

$\phi|\mu = (x \land \text{False}) \lor (\neg x \land \neg \text{False}) = \neg x$

If $\mu = \{y : \text{True}\}$, then

$\phi|\mu = (x \land \text{True}) \lor (\neg x \land \neg \text{True}) = x$
Consider $\exists X. \forall Y. \phi$ where $\phi$ has no quantifiers.

If $Y$ has only a few variables, we can fully expand $\forall Y. \phi$ (i.e., take the conjunction over all assignments):

$$\forall Y. \phi \iff \bigwedge_{\mu \in \text{moves}(Y)} (\phi|\mu) = (\phi|\mu_1) \land \ldots \land (\phi|\mu_n)$$

where $\{\mu_1, \ldots, \mu_n\} = \text{moves}(Y)$.

E.g., $(\exists x. \forall y. x \lor y) \iff \exists x. (x \lor \text{False}) \land (x \lor \text{True})$.

The full expansion can be handed off to a fast SAT solver.

But what if there are many variables in $Y$?
Conjecture: For many practical instances, we only need to consider a small number of moves. Experimental results confirm this.

Let $\omega$ be a subset of moves($Y$).

Partial expansion for $\forall Y. \phi$ over $\omega$ is

$$\bigwedge_{\mu \in \omega} (\phi|_{\mu})$$

- Handicap on Player $\forall$: Can only play moves from $\omega$.
- This partial expansion is called the $\omega$-abstraction of $\forall Y. \phi$. 

CEGAR for 2QBF
CEGAR for 2QBF

To solve $\exists X. \forall Y. \phi$:

0. Initialize $\omega$ to an arbitrary small subset of $\text{moves}(Y)$.

1. Let $\alpha := \bigwedge_{\mu \in \omega} (\phi|_\mu)$
   - The only variables in $\alpha$ are those from $X$ (not $Y$).
   - Handicap on Player $\forall$: Can only play moves from $\omega$.
   - If Player $\exists$ loses $\exists X. \alpha$, then Player $\exists$ also loses $\exists X. \forall Y. \phi$.

2. Try to find $\text{cand} \in \text{moves}(X)$ such that Player $\exists$ wins $\alpha|_{\text{cand}}$.

3. If no such assignment, we’re done: Player $\forall$ wins $\exists X. \forall Y. \phi$.

4. Try to find $\text{cex} \in \text{moves}(Y)$ such that Player $\forall$ wins $(\phi|_{\text{cand}})|_{\text{cex}}$.

5. If no such assignment, we’re done: Player $\exists$ wins $\exists X. \forall Y. \phi$.

6. Let $\omega := \omega \cup \{\text{cex}\}$ and go back to Step 1.
CEGAR for General QBF

- Previous slides dealt with QBF with only two quantifier blocks.
- What about problems with more than two?
- We use a recursive approach.
Full expansion algorithm for QBF

Specification:
NaiveSolve($QX.\phi$) returns a winning move for $X$ (or NULL if none exists), provided that $QX.\phi$ is a closed QBF in strictly-alternating prenex form.

Algorithm 1: NaiveSolve

1. fun NaiveSolve($\exists X.\phi$) = SAT($\phi$) where $\phi$ has no quantifiers.
2. | NaiveSolve($\forall X.\phi$) = SAT($\neg\phi$) where $\phi$ has no quantifiers.
3. | NaiveSolve($\exists X.\forall Y.\phi$) = NaiveSolve(Prenex($\exists X.\bigwedge_{\mu\in moves(Y)}\phi|_{\mu}$))
4. | NaiveSolve($\forall X.\exists Y.\phi$) = NaiveSolve(Prenex($\forall X.\bigvee_{\mu\in moves(Y)}\phi|_{\mu}$))
5. /* Recursion is well-founded; number of quantifiers decreases. */

Notation: We write "$\bar{Q}$" for the opposite type of quantifer as $Q$. That is, if $Q$ is $\exists$ then $\bar{Q}$ is $\forall$, and if $Q$ is $\forall$ then $\bar{Q}$ is $\exists$. 
Abstraction: Partial Expansion

- Algorithm 1 always takes exponential time.
  - Always fully expands each quantifier (except the outermost).
- Instead of fully expanding $\overline{Q}Y.\Phi$, we will partially expand it.
- The partial expansion is considered an abstraction of $\overline{Q}Y.\Phi$.
- Let $\omega$ be a set of assignments to $Y$; i.e., $\omega \subseteq moves(Y)$.
- The $\omega$-abstraction of $\forall Y.\Phi$ is $\bigwedge_{\mu \in \omega} \Phi|\mu$.
- The $\omega$-abstraction of $\exists Y.\Phi$ is $\bigvee_{\mu \in \omega} \Phi|\mu$.
- If $\overline{Q}$ wins an abstraction of $\overline{Q}Y.\Phi$, then $\overline{Q}$ also wins $\overline{Q}Y.\Phi$.
  - $(\left( \bigwedge_{\mu \in \omega} \Phi|\mu \right) \leftrightarrow \text{False}) \Rightarrow (\left( \bigwedge_{\mu \in \text{moves}(Y)} \Phi|\mu \right) \leftrightarrow \text{False})$
  - $(\left( \bigvee_{\mu \in \omega} \Phi|\mu \right) \leftrightarrow \text{True}) \Rightarrow (\left( \bigvee_{\mu \in \text{moves}(Y)} \Phi|\mu \right) \leftrightarrow \text{True})$
Refinement of Abstractions

\[ \alpha_0 \]

\[ \alpha_1 \]

\[ \alpha_n \]

- cand_1
- cand_2
- cand_n

Winning Moves
CEGAR Approach

To solve $QX \cdot \overline{QY} \cdot \Phi$:

0. Initialize $\omega$ such that $\omega \subseteq \text{moves}(Y)$. (We use $\omega = \emptyset$.)

1. Let $\alpha$ be $\omega$-abstraction of $\overline{QY} \cdot \Phi$.
   - Handicap on Player $\overline{Q}$: Can only play moves from $\omega$.
   - If Player $Q$ loses $QX \cdot \alpha$, then Player $Q$ also loses $QX \cdot \overline{QY} \cdot \Phi$.

2. Try to find $\text{cand} \in \text{moves}(X)$ such that Player $Q$ wins $\alpha|_{\text{cand}}$.

3. If no such assignment, we’re done: Player $\overline{Q}$ wins $QX \cdot \overline{QY} \cdot \Phi$.

4. Try to find $\text{cex} \in \text{moves}(Y)$ such that Player $\overline{Q}$ wins $(\Phi|_{\text{cand}})|_{\text{cex}}$.

5. If no such assignment, we’re done: Player $Q$ wins $QX \cdot \overline{QY} \cdot \Phi$.

6. Let $\omega := \omega \cup \{\text{cex}\}$ and go back to Step 1.

Terminates? Yes, because no candidate or counterexample repeats.
Algorithm 2: Basic recursive CEGAR algorithm for QBF

1. **Function** `Solve (QX. Y. Φ)`
2. /* Return value: A winning assignment for X if there is one, NULL otherwise.
3. begin
4. if (Y = ∅) then return (Q=∃ ? SAT(φ) : SAT(¬φ))
5. ω := ∅
6. while true do
7. \[ \alpha := \begin{cases} \exists X. \bigwedge_{\mu \in \omega} \Phi|_{\mu} & \text{if } Q = ∀ \\
\forall X. \bigvee_{\mu \in \omega} \Phi|_{\mu} & \text{if } Q = \exists \end{cases} \]
8. `cand := Solve(Prenex(α))` // find a candidate solution
9. if `cand` = NULL then return NULL
10. Remove from `cand` any variables not in X
11. `cex := Solve(\bar{Q}Y. Φ|_{cand})` // find a counterexample
12. if `cex` = NULL then return `cand`
13. ω := ω ∪ {cex}
14. end
15. end
Optimizations

- Prenexing the formula can be harmful.
  - Loses information about dependencies.

- Technique to avoid re-prenexing formula.
  - But input is still originally in prenex form (QDIMACS).

- Incremental interface to MiniSAT.

- Our final implementation:
  - RAReQS: Recursive Abstraction Refinement QBF Solver

- Notation: $(\exists X. \{\Phi_1, \ldots, \Phi_n\}) = \exists X.(\Phi_1 \land \ldots \land \Phi_n)$

- Notation: $(\forall X. \{\Phi_1, \ldots, \Phi_n\}) = \forall X.(\Phi_1 \lor \ldots \lor \Phi_n)$
RAREQS: Optimized recursive CEGAR algorithm

Algorithm 3: RAREQS

1 Function RAREQS (QX. {Φ₁, . . . , Φₙ})
2 /* Return value: A winning assignment for X if there is one, NULL otherwise.
3 begin
4 if (Φᵢ have no quantifiers) then return Q=∃ ? SAT(∧ᵢ Φᵢ) : SAT(¬(∨ᵢ Φ))
5 α := QX. {}  
6 while true do
7     cand := RAREQS(α)  // find a candidate solution
8     if cand = NULL then return NULL
9     Remove from cand any variables not in X.
10    for i := 1 to n do  cexᵢ := RAREQS(Φᵢ|cand)  // find a counterexample
11    if cexᵢ = NULL for all i ∈ {1..n} then return cand
12    let l ∈ {1..n} be such that  cexₗ ≠ NULL
13    α := Refine(α, Φₗ|cexₗ)
14 end
DPLL Integration

- CEGAR learning can also be integrated into a DPLL solver.
- Since conjunction and disjunction are idempotent,

\[
\exists Y. \Phi = (\exists Y. \Phi) \lor \Phi|_\mu, \text{ where } \mu \in \text{moves}(Y)
\]

\[
\forall Y. \Phi = (\forall Y. \Phi) \land \Phi|_\mu, \text{ where } \mu \in \text{moves}(Y)
\]

where \text{moves}(Y) is the set of all assignments to \(Y\).

- Can substitute \(\exists X. \forall Y. \Phi\) with \(\exists X. (\forall Y. \Phi) \land \Phi|_\mu\) where \(\mu \in \text{moves}(Y)\).
- Do this if we find a candidate for \(X\) and a countermove for \(Y\).
- May increase propagation power.
- Implemented in GhostQ for case where \(Y\) is innermost block.
Cactus Plot of Results

Number of instances solved vs per-instance time limit

Benchmarks used: Formal Verification and Planning families from QBFLIB.
Summary of Results

Total instances solved (out of 4669):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>RAReQS</th>
<th>GhostQ</th>
<th>GhostQ-C</th>
<th>Qube</th>
<th>Quantor</th>
<th>Nenofex</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAReQS</td>
<td>3868</td>
<td>2449</td>
<td>2801</td>
<td>2916</td>
<td>1462</td>
<td>1317</td>
</tr>
<tr>
<td>GhostQ-C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qube</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nenofex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RAReQS vs GhostQ
- Only RAReQS: 1661
- Only GhostQ: 242
- Both: 2207

RAReQS vs Qube
- Only RAReQS: 998
- Only Qube: 46
- Both: 2870

RAReQS vs Quantor
- Only RAReQS: 2436
- Only Quantor: 30
- Both: 1432
Conclusion

- Introduced CEGAR-based algorithm for QBF, and implemented a solver **RAReQS** based on it.
- RAReQS builds an abstraction of the given formula by constructing a partial expansion.
- RAReQS performs very well compared to existing solvers.
  - Solves many more instances within per-instance time limit.
  - Can verify more complex industrial hardware designs.
- Future work: Integrate more tightly with DPLL techniques.
  - E.g., non-chronological backtracking between quantifier blocks.