Extending DPLL-Based QBF Solvers to Handle Free Variables

Will Klieber, Mikoláš Janota,
Joao Marques-Silva, Edmund Clarke

July 9, 2013

Open QBF

- Closed QBF: All variables quantified; answer is True or False.
- Open QBF: Contains free (unquantified) variables.
- Goal: Find equivalent propositional formula.
- E.g., given $\exists x.\ x \wedge (y \vee z)$, return $y \vee z$.

Open QBF

- Closed QBF: All variables quantified; answer is True or False.
- Open QBF: Contains free (unquantified) variables.
- Goal: Find equivalent propositional formula.
- E.g., given $\exists x.\, x \wedge (y \vee z)$, return $y \vee z$.
- Applications: symbolic MC, synthesis from formal spec, etc.

Outline

- Naïve Algorithm
- Introduce *sequents* that generalize clauses for open QBF in CNF (without ghost variables)
- Experimental results
- *Ghost variables*: see paper.

Naïve Algorithm

- Notation: "$\mathrm{ite}(x, \phi_1, \phi_2)$" is a formula with an *if-then-else*:

$$\mathrm{ite}(x, \phi_1, \phi_2) \;=\; (x \wedge \phi_1) \vee (\neg x \wedge \phi_2)$$

Naïve Algorithm

- Notation: "$\mathrm{ite}(x, \phi_1, \phi_2)$" is a formula with an *if-then-else*:

$$\mathrm{ite}(x, \phi_1, \phi_2) \;=\; (x \wedge \phi_1) \vee (\neg x \wedge \phi_2)$$

- Recursively Shannon-expand on free variables:

$$\Phi \;=\; \mathrm{ite}(x,\ \Phi|_{x=\mathsf{True}},\ \Phi|_{x=\mathsf{False}})$$

Naïve Algorithm

- Notation: "$\text{ite}(x, \phi_1, \phi_2)$" is a formula with an *if-then-else*:

$$\text{ite}(x, \phi_1, \phi_2) \;=\; (x \wedge \phi_1) \vee (\neg x \wedge \phi_2)$$

- Recursively Shannon-expand on free variables:

$$\Phi \;=\; \text{ite}(x,\; \Phi|_{x=\textsf{True}},\; \Phi|_{x=\textsf{False}})$$

- Base case (no more free variables): Give to closed-QBF solver.

Naïve Algorithm

```
function solve(Φ) {
    if (Φ has no free variables)
        return closed_qbf_solve(Φ);



}
```

Naïve Algorithm

```
function solve(Φ) {
    if (Φ has no free variables)
        return closed_qbf_solve(Φ);
    x := (a free variable in Φ);
    return ite(x, solve( Φ|x=True ),
                  solve( Φ|x=False ));
}
```

Naïve Algorithm

```
function solve(Φ) {
    if (Φ has no free variables)
        return closed_qbf_solve(Φ);
    x := (a free variable in Φ);
    return ite(x, solve( Φ|x=True ),
                  solve( Φ|x=False ));
}
```

Builds OBDD if:

1. same branch order,
2. formula construction is memoized, and
3. $\mathrm{ite}(x,\ \phi,\ \phi)$ is simplified to ϕ.

Naïve Algorithm

- Naïve Algorithm:
  - Similar to DPLL in terms of branching.
  - But lacks many optimizations that make DPLL fast:
    - Non-chronological backtracking
    - Clause learning
- Our open-QBF technique:
  - Extend existing closed-QBF algorithm to allow free variables.

Preliminaries

- **Prenex Form:** $Q_1\vec{x}_1 ... Q_n\vec{x}_n.\, \phi$ where ϕ has no quantifiers.

Preliminaries

- **Prenex Form:** $Q_1\vec{x}_1 ... Q_n\vec{x}_n.\, \phi$ where ϕ has no quantifiers.
- In $\forall x.\exists y.\, \phi$, we say that y is **downstream** of x.
  - $\exists y$ occurs inside scope of $\forall x$.

Preliminaries

- **Prenex Form:** $Q_1\vec{x}_1 ... Q_n\vec{x}_n.\, \phi$ where ϕ has no quantifiers.
- In $\forall x.\exists y.\, \phi$, we say that y is **downstream** of x.
  - $\exists y$ occurs inside scope of $\forall x$.
- Free variables are upstream of all quantified variables.

Preliminaries

- **Prenex Form:** $Q_1\vec{x}_1 ... Q_n\vec{x}_n.\, \phi$ where ϕ has no quantifiers.
- In $\forall x.\exists y.\, \phi$, we say that y is **downstream** of x.
  - $\exists y$ occurs inside scope of $\forall x$.
- Free variables are upstream of all quantified variables.
- We identify assignment π with the set of literals made true by π.
- E.g., identify $\{(e_1, \text{True}), (u_2, \text{False})\}$ with $\{e_1, \neg u_2\}$.

Preliminaries

- **Prenex Form:** $Q_1\vec{x}_1...Q_n\vec{x}_n.\,\phi$ where ϕ has no quantifiers.
- In $\forall x.\exists y.\,\phi$, we say that y is **downstream** of x.
  - $\exists y$ occurs inside scope of $\forall x$.
- Free variables are upstream of all quantified variables.
- We identify assignment π with the set of literals made true by π.
- E.g., identify $\{(e_1, \text{True}), (u_2, \text{False})\}$ with $\{e_1, \neg u_2\}$.
- **Substitution:** $\Phi|_\pi$ substitutes assigned variables with values (even if bound by quantifier, which gets deleted).

QBF as a Game

- Existential variables are **owned** by Player $\exists$.
- Universal variables are **owned** by Player $\forall$.
- Players assign variables in quantification order.
- The **goal** of Player $\exists$ is to make Φ be true.
- The **goal** of Player $\forall$ is to make Φ be false.

Properties of Clauses and Cubes

- Motivate definition of sequents.
- If π falsifies all literals in clause C in CNF Φ, then $\Phi|_\pi = \text{False}$.

Properties of Clauses and Cubes

- Motivate definition of sequents.
- If π falsifies all literals in clause C in CNF Φ, then $\Phi|_\pi = \text{False}$.
- If π falsifies all existential literals in clause C in CNF Φ and doesn't satisfy any universal literals in C, then $\Phi|_\pi = \text{False}$.

Properties of Clauses and Cubes

- Motivate definition of sequents.
- If π falsifies all literals in clause C in CNF Φ, then $\Phi|_\pi =$ False.
- If π falsifies all existential literals in clause C in CNF Φ and doesn't satisfy any universal literals in C, then $\Phi|_\pi =$ False.
- If π satisfies all universal literals in a cube C in a DNF Φ and doesn't falsify any existential literals in C, then $\Phi|_\pi =$ True.

Properties of Clauses and Cubes

- Motivate definition of sequents.
- If π falsifies all literals in clause C in CNF Φ, then $\Phi|_\pi = \text{False}$.
- If π falsifies all existential literals in clause C in CNF Φ and doesn't satisfy any universal literals in C, then $\Phi|_\pi = \text{False}$.
- If π satisfies all universal literals in a cube C in a DNF Φ and doesn't falsify any existential literals in C, then $\Phi|_\pi = \text{True}$.
- Tautological clauses learned via long-distance resolution? (Assuming $\forall$-reduction is done only on-the-fly, during unit prop.)

$\langle L^{\text{now}}, L^{\text{fut}} \rangle$ Sequents

- **Definition.** A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}.
- **Definition.** We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** assignment π iff:
  1. for every literal ℓ in L^{now}, $\ell|_\pi = \text{True}$, and
  2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \text{True}$ or $\ell \notin \text{vars}(\pi)$.

$\langle L^{\text{now}}, L^{\text{fut}} \rangle$ Sequents

- **Definition.** A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}.
- **Definition.** We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** assignment π iff:
  1. for every literal ℓ in L^{now}, $\ell|_\pi = \text{True}$, and
  2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \text{True}$ or $\ell \notin \text{vars}(\pi)$.
- E.g., $\langle \{e\}, \{u\} \rangle$ matches $\{e\}$ and $\{e, u\}$,

$\langle L^{\text{now}}, L^{\text{fut}} \rangle$ Sequents

- **Definition.** A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}.
- **Definition.** We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** assignment π iff:
  1. for every literal ℓ in L^{now}, $\ell|_\pi = \text{True}$, and
  2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \text{True}$ or $\ell \notin \text{vars}(\pi)$.
- E.g., $\langle \{e\}, \{u\} \rangle$ matches $\{e\}$ and $\{e, u\}$, but does not match $\{\}$ or $\{e, \neg u\}$.

$\langle L^{\text{now}}, L^{\text{fut}} \rangle$ Sequents

- **Definition.** A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}.
- **Definition.** We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** assignment π iff:
  1. for every literal ℓ in L^{now}, $\ell|_\pi = \text{True}$, and
  2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \text{True}$ or $\ell \notin \text{vars}(\pi)$.
- E.g., $\langle \{e\}, \{u\} \rangle$ matches $\{e\}$ and $\{e, u\}$, but does not match $\{\}$ or $\{e, \neg u\}$.
- $\langle L^{\text{now}}, \{\ell, \neg\ell\} \rangle$ matches π only if π doesn't assign ℓ.

$\langle L^{\text{now}}, L^{\text{fut}} \rangle$ Sequents

- **Definition.** A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}.
- **Definition.** We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** assignment π iff:
  1. for every literal ℓ in L^{now}, $\ell|_\pi = \text{True}$, and
  2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \text{True}$ or $\ell \notin \text{vars}(\pi)$.
- **Definition.** "$\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (\Phi \Leftrightarrow \psi)$" means "for all assignments π that match $\langle L^{\text{now}}, L^{\text{fut}} \rangle$, $\Phi|_\pi$ is logically equivalent to $\psi|_\pi$ unless π is a **don't-care** assignment".

$\langle L^{\text{now}}, L^{\text{fut}} \rangle$ Sequents

- **Definition.** A **game-state specifier** is a pair $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ consisting of two sets of literals, L^{now} and L^{fut}.
- **Definition.** We say that $\langle L^{\text{now}}, L^{\text{fut}} \rangle$ **matches** assignment π iff:
  1. for every literal ℓ in L^{now}, $\ell|_\pi = \text{True}$, and
  2. for every literal ℓ in L^{fut}, either $\ell|_\pi = \text{True}$ or $\ell \notin \text{vars}(\pi)$.
- **Definition.** "$\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (\Phi \Leftrightarrow \psi)$" means "for all assignments π that match $\langle L^{\text{now}}, L^{\text{fut}} \rangle$, $\Phi|_\pi$ is logically equivalent to $\psi|_\pi$ unless π is a **don't-care** assignment".
- Without ghost literals: No assignments are don't-care.
- With ghost literals: Some assignments are don't-care.

Correspondence of Sequents to Clauses and Cubes

- Consider a QBF with existential literals $e_1 \dots e_n$ and universal literals $u_1 \dots u_m$.
- Clause $(e_1 \vee \dots \vee e_n \vee u_1 \vee \dots \vee u_m)$ in CNF Φ_{in} corresponds to sequent $\langle\{\neg e_1, \dots, \neg e_n\}, \{\neg u_1, \dots, \neg u_m\}\rangle \models (\Phi_{in} \Leftrightarrow \text{False})$.

Correspondence of Sequents to Clauses and Cubes

- Consider a QBF with existential literals $e_1 \dots e_n$ and universal literals $u_1 \dots u_m$.
- Clause $(e_1 \vee \dots \vee e_n \vee u_1 \vee \dots \vee u_m)$ in CNF Φ_{in} corresponds to sequent $\langle \{\neg e_1, \dots, \neg e_n\}, \{\neg u_1, \dots, \neg u_m\} \rangle \models (\Phi_{in} \Leftrightarrow \text{False})$.
- Cube $(e_1 \wedge \dots \wedge e_n \wedge u_1 \wedge \dots \wedge u_m)$ in DNF Φ_{in} corresponds to sequent $\langle \{u_1, \dots, u_m\}, \{e_1, \dots, e_n\} \rangle \models (\Phi_{in} \Leftrightarrow \text{True})$.

Correspondence of Sequents to Clauses and Cubes

- Consider a QBF with existential literals $e_1 \dots e_n$ and universal literals $u_1 \dots u_m$.
- Clause $(e_1 \vee \dots \vee e_n \vee u_1 \vee \dots \vee u_m)$ in CNF Φ_{in} corresponds to sequent $\langle\{\neg e_1, \dots, \neg e_n\}, \{\neg u_1, \dots, \neg u_m\}\rangle \models (\Phi_{in} \Leftrightarrow \text{False})$.
- Cube $(e_1 \wedge \dots \wedge e_n \wedge u_1 \wedge \dots \wedge u_m)$ in DNF Φ_{in} corresponds to sequent $\langle\{u_1, \dots, u_m\}, \{e_1, \dots, e_n\}\rangle \models (\Phi_{in} \Leftrightarrow \text{True})$.
- Sequents generalize clauses/cubes because $\langle L^{\text{now}}, L^{\text{fut}}\rangle \models (\Phi \Leftrightarrow \psi)$ can have ψ be a formula in terms of free variables.

Alternate Sequent Notation

- “$\langle L^{\mathsf{now}}, L^{\mathsf{fut}} \rangle \models (\exists$ loses $\Phi)$” means “$\langle L^{\mathsf{now}}, L^{\mathsf{fut}} \rangle \models (\Phi \Leftrightarrow$ False$)$”.

- “$\langle L^{\mathsf{now}}, L^{\mathsf{fut}} \rangle \models (\forall$ loses $\Phi)$” means “$\langle L^{\mathsf{now}}, L^{\mathsf{fut}} \rangle \models (\Phi \Leftrightarrow$ True$)$”.

Resolution rule for free variable

Literal r is free

$$\frac{\langle L_1^{\text{now}} \cup \{r\},\, L_1^{\text{fut}}\rangle \models (\Phi_{in} \Leftrightarrow \psi_1) \qquad \langle L_2^{\text{now}} \cup \{\neg r\},\, L_2^{\text{fut}}\rangle \models (\Phi_{in} \Leftrightarrow \psi_2)}{\langle L_1^{\text{now}} \cup L_2^{\text{now}},\, L_1^{\text{fut}} \cup L_2^{\text{fut}} \cup \{r, \neg r\}\rangle \models (\Phi_{in} \Leftrightarrow \text{ite}(r, \psi_1, \psi_2))}$$

Top-level algorithm

```
initialize_sequent_database();
π_cur := ∅;  Propagate();
while (true) {



}
```

Top-level algorithm

```
initialize_sequent_database();
π_cur := ∅;  Propagate();
while (true) {
  while (π_cur doesn't match any database sequent) {
    DecideLit();
    Propagate();
  }

}
```

Top-level algorithm

`initialize_sequent_database();`
π_{cur} `:=` $\varnothing$`;` `Propagate();`
`while (true) {`
  `while (`π_{cur} doesn't match any database sequent`) {`
    `DecideLit();`
    `Propagate();`
  `}`
  `Learn();`
  `if` (learned seq has form $\langle \varnothing, \ L^{\texttt{fut}} \rangle \models (\Phi_{in} \Leftrightarrow \psi)$) `return` ψ`;`
  `Backtrack();`
  `Propagate();`
`}`

Propagation

- Let *seq* be a sequent $\langle L^{\mathsf{now}}, L^{\mathsf{fut}} \rangle \models (\Phi_{in} \Leftrightarrow \psi)$ in database.
- If there is a literal $\ell \in L^{\mathsf{now}}$ such that
    1. $\pi_{cur} \cup \{\ell\}$ matches *seq*, and
    2. ℓ is not downstream of any unassigned literals in L^{fut},

  then $\neg\ell$ is *forced*; it is added to the current assignment π_{cur}.

Propagation

- Let *seq* be a sequent $\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (\Phi_{in} \Leftrightarrow \psi)$ in database.
- If there is a literal $\ell \in L^{\text{now}}$ such that
  1. $\pi_{cur} \cup \{\ell\}$ matches *seq*, and
  2. ℓ is not downstream of any unassigned literals in L^{fut},

  then $\neg\ell$ is *forced*; it is added to the current assignment π_{cur}.
- Propagation ensures that the solver never re-explores areas of the search space for which it already knows the answer.

Learning

```
func Learn() {
    seq := (the database sequent that matches π_cur);
    while (true) {

    }
}
```

Learning

```
func Learn() {
    seq := (the database sequent that matches π_cur);
    while (true) {
        r := (the most recently assigned literal in seq.L^now)
        seq := Resolve(seq, antecedent[r]);



    }
}
```

Learning

```
func Learn() {
    seq := (the database sequent that matches π_cur);
    while (true) {
        r := (the most recently assigned literal in seq.L^now)
        seq := Resolve(seq, antecedent[r]);
        if (seq.L^now = ∅ or has_good_UIP(seq))
            return seq;
    }
}
```

Resolution rule for quantified variable (case 1)

$$\frac{\begin{array}{l}\text{The quantifier type of } r \text{ in } \Phi \text{ is } Q \\ \langle L_1^{\text{now}} \cup \{r\},\, L_1^{\text{fut}} \rangle \models (Q \text{ loses } \Phi_{in}) \\ \langle L_2^{\text{now}} \cup \{\neg r\},\, L_2^{\text{fut}} \rangle \models (Q \text{ loses } \Phi_{in}) \\ \text{Opponent of } Q \text{ owns all literals in } L_1^{\text{fut}} \\ r \text{ is not downstream of any } \ell \text{ such that } \ell \in L_1^{\text{fut}} \text{ and } \neg\ell \in (L_1^{\text{fut}} \cup L_2^{\text{fut}})\end{array}}{\langle L_1^{\text{now}} \cup L_2^{\text{now}},\, L_1^{\text{fut}} \cup L_2^{\text{fut}} \rangle \models (Q \text{ loses } \Phi_{in})}$$

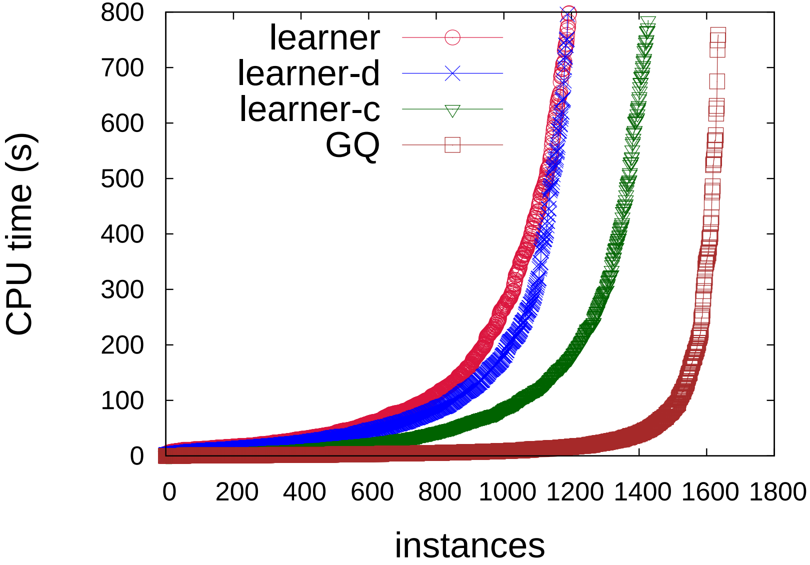
Resolution rule for quantified variable (case 2)

$$\frac{\begin{array}{l}\text{The quantifier type of } r \text{ in } \Phi \text{ is } Q \\ \langle L_1^{\mathsf{now}} \cup \{r\},\, L_1^{\mathsf{fut}} \rangle \models (Q \text{ loses } \Phi_{in}) \\ \langle L_2^{\mathsf{now}} \cup \{\neg r\},\, L_2^{\mathsf{fut}} \rangle \models (\Phi_{in} \Leftrightarrow \psi) \\ \text{Opponent of } Q \text{ owns all literals in } L_1^{\mathsf{fut}} \\ r \text{ is not downstream of any } \ell \text{ such that } \ell \in L_1^{\mathsf{fut}} \text{ and } \neg\ell \in (L_1^{\mathsf{fut}} \cup L_2^{\mathsf{fut}})\end{array}}{\langle L_1^{\mathsf{now}} \cup L_2^{\mathsf{now}},\, L_1^{\mathsf{fut}} \cup L_2^{\mathsf{fut}} \cup \{\neg r\} \rangle \models (\Phi_{in} \Leftrightarrow \psi)}$$

Experimental Comparison

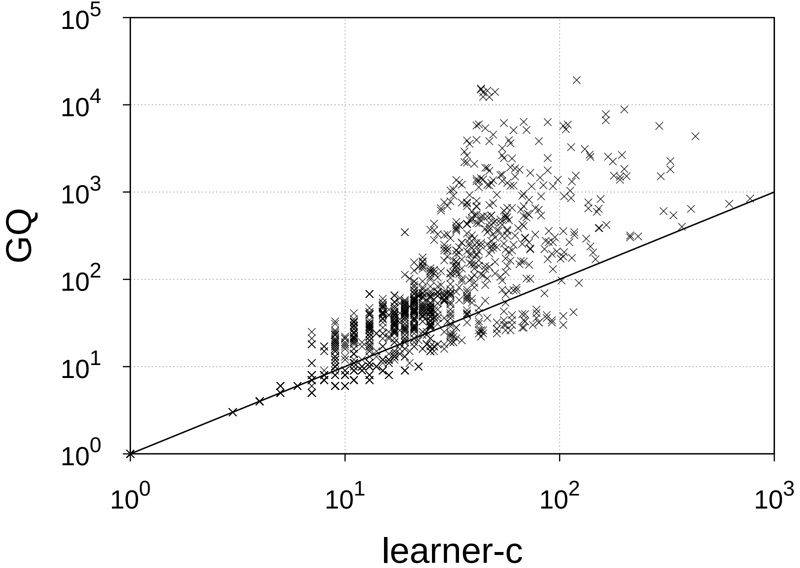
- Our solver: GhostQ.
- Compared to computational-learning solver from:
  B. Becker, R. Ehlers, M. Lewis, and P. Marin,
  "ALLQBF solving by computational learning" (ATVA 2012).
- Benchmarks (from same paper): synthesis from formal specifications.

Cactus Plot



CPU time (s)
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000
1200
1400
1600
1800
instances
learner
learner-d
learner-c
GQ

Formula Size



Conclusion

- DPLL-based solver for open QBF.
- Sequents generalize clauses and cubes.
- Generates proof certificates.
- Our solver produces **unordered** BDDs.
  - Unordered because of unit propagation.
  - In our experience, often larger than OBDDs.
- More details: preprint of CP 2013 paper on Will Klieber's website.