A Non-Prenex, Non-Clausal QBF Solver with Game-State Learning

Will Klieber, Samir Sapra, Sicun Gao, Edmund Clarke
Carnegie Mellon University

July 13, 2010

Preview

- Non-prenex, non-clausal QBF solver (DPLL-based).
- **Game-state learning**
  - Reformulation of clause/cube learning,
    extended to non-prenex case.
- **Ghost literals**
  - Symmetric propagation technique,
    exploits structure of non-prenex, non-clausal instances.

Why study QBF?

- Practical problems naturally expressed in QBF.
- Formal verification: e.g., Bounded Model Checking
- SAT solvers: success in formal verification.
  - Hopefully QBF solvers too.

Semantics

- $\boxed{\phi|_{x=\mathtt{T}}}$: plug in $\mathtt{T}$ (true) for x. E.g., $(x \vee y)|_{x=\mathtt{T}} = (\mathtt{T} \vee y) = \mathtt{T}$.
- $[\forall x.\, \phi] = [\phi|_{x=\mathtt{T}}] \wedge [\phi|_{x=\mathtt{F}}]$ (universal quantifier)
- $[\exists x.\, \phi] = [\phi|_{x=\mathtt{T}}] \vee [\phi|_{x=\mathtt{F}}]$ (existential quantifier)

QBF Solver:

- Input formula: InFmla
- Assume each variable quantified exactly once in InFmla.
  - No free variables.
  - InFmla evaluates to either $\mathtt{T}$ or $\mathtt{F}$.
- Goal: determine the truth value of InFmla.

QBF as a Game

- Existential variables are owned by Player `E`.
  Universal variables are owned by Player `U`.
- Players assign variables in quantification order.
  - Start with outermost quantified (leftmost).
- Player `E`'s goal: Make *InFmla* be true.
  Player `U`'s goal: Make *InFmla* be false.
- To make this more precise: *reduction* (next slide).

Reduction of a Formula

- Let "π" denote a (partial) assignment of values to variables.
- To construct the *reduction* of f under π (denoted "$f|\pi$"):
  - For each variable x in π:
    - Delete quantifier of x.
    - Replace occurrences with assigned value.

Reduction of a Formula

- Let "π" denote a (partial) assignment of values to variables.
- To construct the *reduction* of f under π (denoted "$f|\pi$"):
  - For each variable x in π:
    - Delete quantifier of x.
    - Replace occurrences with assigned value.
- Example:
  - $f = (\exists e_1. \forall u_2.\ e_1 \wedge u_2)$, $\pi = \{e_1 : \texttt{True}\}$
  - Reduction: $f|\pi = (\forall u_2.\ \texttt{True} \wedge u_2)$
- We say "P wins f under π" iff P has a winning strategy for $f|\pi$.
- Player E wins f under π iff $f|\pi$ is true.
- Player U wins f under π iff $f|\pi$ is false.

Quantification Order

- Don't need strict outer-to-inner.
- Block of one type of quantifier.
  $\exists e_1 \exists e_2 \exists e_3 \forall u_4 \forall u_5 . f$
- We say $\{e_1, e_2, e_3\}$ are *ready*, while $\{u_4, u_5\}$ are *unready* (under the empty assignment).

Quantification Order

- Don't need strict outer-to-inner.
- Block of one type of quantifier.
  $\exists e_1 \exists e_2 \exists e_3 \forall u_4 \forall u_5 . f$
- We say $\{e_1, e_2, e_3\}$ are *ready*, while $\{u_4, u_5\}$ are *unready* (under the empty assignment).
- **Definition:** An unassigned variable is *ready* iff its quantifier is not within the scope of the quantifier of an unassigned variable owned by the opposing player.
- E.g., $\exists e_4 . ((\exists e_5 . f) \wedge (\forall u_6 . h))$
  - e_4 and e_5 are ready, while u_6 is unready.

Representation of Formulas

- Negation-Normal Form (NNF)
  - Logical operators: AND, OR, NOT.
  - Negations are pushed inward by De Morgan's; occur only in front of variables.
  - Literal: a variable or its negation.
- Prenex: All quantifiers at beginning.
  $\underbrace{\forall x \exists y \forall z.}_{\text{prefix}} \underbrace{((x \wedge y) \vee (y \wedge z))}_{\text{matrix}}$
- Early QBF solvers: Prenex CNF (Conjunctive Normal Form)
- Prenexing is harmful (since it limits the branching order).
- Converting to CNF is harmful (since Player E's variables are conflated with gate variables).

Representation of Formulas (cont.)

- **Gate variables:** label each conjunction/disjunction.
- **Prime gate vars:** include quantifier prefix.
- **Input variables:** original (non-gate) variables.

$$\exists e_{10}\,[\underbrace{[\exists e_{11}\,\forall u_{21}\,\overbrace{(e_{10}\wedge e_{11}\wedge u_{21})}^{g_1}]}_{g_1'}\wedge\underbrace{[\forall u_{22}\,\exists e_{30}\,\overbrace{(e_{10}\wedge u_{22}\wedge e_{30})}^{g_2}]}_{g_2'}]$$

Representation of Formulas (cont.)

- **Gate variables:** label each conjunction/disjunction.
- **Prime gate vars:** include quantifier prefix.
- **Input variables:** original (non-gate) variables.

$$\exists e_{10}\, [\underbrace{[\exists e_{11}\, \forall u_{21}\, \overbrace{(e_{10} \wedge e_{11} \wedge u_{21})}^{g_1}]}_{g_1'} \wedge \underbrace{[\forall u_{22}\, \exists e_{30}\, \overbrace{(e_{10} \wedge u_{22} \wedge e_{30})}^{g_2}]}_{g_2'}]$$

- Quantified subformulas (e.g., g_1', g_2'): subgames.
- Subgames g_1' and g_2' are independent after e_{10} assigned.

- Implementation: Pure NNF is not required.
  A quantifier-free subformula can be represented in circuit form.

Representation of Current Assignment

- During solving process, we assign values to the input variables.
- We write "*CurAsgn*" to denote the current assignment.
- *CurAsgn* may be represented by the set of literals assigned true.
- E.g., $\{e_1{=}\mathtt{T},\ e_2{=}\mathtt{F}\}$ may be represented by $\{e_1, \neg e_2\}$.

Top-level algorithm

```
/* Goal: Find out who wins InFmla (under empty asgn). */
while (true) {
  while (don't know who wins InFmla under CurAsgn) {
    DecideLit();   // Pick a ready literal.
    Propagate();   // Detect forced literals.
  }
  ...
  ...
  ...
  ...
}
```

Top-level algorithm

```
/* Goal: Find out who wins InFmla (under empty asgn). */
while (true) {
  while (don't know who wins InFmla under CurAsgn) {
    DecideLit();   // Pick a ready literal.
    Propagate();   // Detect forced literals.
  }
  Learn so that we don't repeat same decisions again;
  if (we learned who wins InFmla under ∅) return;
  Backtrack(); // Remove recent literals from CurAsgn;
  Propagate(); // Learned information will force a literal.
}
```

Optional modification: Target in on a subgame when independent.

Game-State Learning – Motivation

- Reformulation of clause/cube learning, extended to non-prenex.
- For prenex CNF: merely cosmetic differences between game-state learning and clause/cube learning.

$$\exists e_1 \exists e_3 \forall u_4 \exists e_5 \exists e_7 . \underbrace{(e_1 \vee e_3 \vee u_4 \vee e_5)}_{g_1} \wedge \underbrace{(e_1 \vee \neg e_3 \vee \neg u_4 \vee e_7)}_{g_2} \wedge \ldots$$

- g_1: If $\{e_1, e_3, u_4, e_5\}$ are false, then U wins.

Game-State Learning – Motivation

- Reformulation of clause/cube learning, extended to non-prenex.
- For prenex CNF: merely cosmetic differences between game-state learning and clause/cube learning.

$$\exists e_1 \exists e_3 \forall u_4 \exists e_5 \exists e_7 . \underbrace{(e_1 \vee e_3 \vee u_4 \vee e_5)}_{g_1} \wedge \underbrace{(e_1 \vee \neg e_3 \vee \neg u_4 \vee e_7)}_{g_2} \wedge \ldots$$

- g_1: If $\{e_1, e_3, u_4, e_5\}$ are false, then U wins.
- g_1: If $\{\neg e_1, \neg e_3, \neg u_4, \neg e_5\}$ are true, then U wins.

Game-State Learning – Motivation

- Reformulation of clause/cube learning, extended to non-prenex.
- For prenex CNF: merely cosmetic differences between game-state learning and clause/cube learning.

$$\exists e_1 \exists e_3 \forall u_4 \exists e_5 \exists e_7 . \underbrace{(e_1 \vee e_3 \vee u_4 \vee e_5)}_{g_1} \wedge \underbrace{(e_1 \vee \neg e_3 \vee \neg u_4 \vee e_7)}_{g_2} \wedge \ldots$$

- g_1: If $\{e_1, e_3, u_4, e_5\}$ are false, then U wins.
- g_1: If $\{\neg e_1, \neg e_3, \neg u_4, \neg e_5\}$ are true, then U wins.
- g_1: If $\{\neg e_1, \neg e_3, \neg e_5\}$ are true and $\neg u_4$ is non-false, then U wins. ("non-false": "true or unassigned")

Game-State Learning – Motivation

- Reformulation of clause/cube learning, extended to non-prenex.
- For prenex CNF: merely cosmetic differences between game-state learning and clause/cube learning.

$$\exists e_1 \exists e_3 \forall u_4 \exists e_5 \exists e_7. \underbrace{(e_1 \vee e_3 \vee u_4 \vee e_5)}_{g_1} \wedge \underbrace{(e_1 \vee \neg e_3 \vee \neg u_4 \vee e_7)}_{g_2} \wedge \ldots$$

- g_1: If $\{e_1, e_3, u_4, e_5\}$ are false, then U wins.
- g_1: If $\{\neg e_1, \neg e_3, \neg u_4, \neg e_5\}$ are true, then U wins.
- g_1: If $\{\neg e_1, \neg e_3, \neg e_5\}$ are true and $\neg u_4$ is non-false, then U wins. ("non-false": "true or unassigned")
- Game-state sequent: "$\langle \{\neg e_1, \neg e_3, \neg e_5\}, \{\neg u_4\} \rangle \models (\mathtt{U}\ \text{wins}\ \mathit{InFmla})$"
- Can learn who wins a subgame.

Game-State Sequents

- Consider a subgame f (a quantified subformula).
- "$\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (P \text{ wins } f)$" means "Player P wins f whenever:
  1. every literal in L^{now} is true, and
  2. every literal in L^{fut} is non-false (i.e., true or unassigned) (i.e., every literal in L^{fut} can be true in the future)."

Game-State Sequents

- Consider a subgame f (a quantified subformula).
- "$\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (P \text{ wins } f)$" means "Player P wins f whenever:
  1. every literal in L^{now} is true, and
  2. every literal in L^{fut} is non-false (i.e., true or unassigned) (i.e., every literal in L^{fut} can be true in the future)."
- L^{now} may contain both input literals and gate literals; L^{fut} may contain only input literals.

Game-State Sequents

- Consider a subgame f (a quantified subformula).
- "$\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (P \text{ wins } f)$" means "Player P wins f whenever:
  1. every literal in L^{now} is true, and
  2. every literal in L^{fut} is non-false (i.e., true or unassigned) (i.e., every literal in L^{fut} can be true in the future)."
- "P wins f whenever ...":
  "P wins f under all assignments meeting the conditions"
  (even if out of quantification order, due to forced literals).
- Player E wins f under π iff $f|_\pi$ is true.
  Player U wins f under π iff $f|_\pi$ is false.

Game-State Sequents

- Consider a subgame f (a quantified subformula).
- $\langle L^{\text{now}}, L^{\text{fut}} \rangle \models (P \text{ wins } f)$ **matches** an assignment π iff, under π,
  1. every literal in L^{now} is true, and
  2. every literal in L^{fut} is non-false (i.e., true or unassigned) (i.e., every literal in L^{fut} can be true in the future)."

Propagation and Learning

- At time t^*: $\mathit{CurAsgn} = \pi^*$, targetted subgame is f.
- Suppose $\pi^* \cup \{\neg\ell\}$ matches $\underbrace{\langle L_B^{\mathsf{now}} \cup \{\neg\ell\}, L_B^{\mathsf{fut}}\rangle \models (P \text{ loses } h)}_{\substack{\text{in game-state database} \\ h \text{ is a subgame of } f}}$.

Propagation and Learning

- At time t^*: $CurAsgn = \pi^*$, targetted subgame is f.
- Suppose $\pi^* \cup \{\neg\ell\}$ matches $\langle L_B^{\text{now}} \cup \{\neg\ell\}, L_B^{\text{fut}} \rangle \models (P \text{ loses } h)$, and
  - ℓ is owned by P.
  - ℓ does not appear outside h (and h is a subgame of f).
  - ℓ is upstream of all literals in L_B^{fut}. (ℓ gets picked before L_B^{fut})
- For P to win f, making $\ell = \mathtt{F}$ is at least as bad as $\ell = \mathtt{T}$.
  - Only way ℓ can help P win f is by helping P win h.
  - If P makes $\ell = \mathtt{F}$, then P loses h.

Propagation and Learning

- At time t^*: $\mathit{CurAsgn} = \pi^*$, targetted subgame is f.
- Suppose $\pi^* \cup \{\neg\ell\}$ matches $\langle L_B^{\text{now}} \cup \{\neg\ell\}, L_B^{\text{fut}}\rangle \models (P \text{ loses } h)$, and
  - ℓ is owned by P.
  - ℓ does not appear outside h (and h is a subgame of f).
  - ℓ is upstream of all literals in L_B^{fut}. (ℓ gets picked before L_B^{fut})
- For P to win f, making $\ell = \mathtt{F}$ is at least as bad as $\ell = \mathtt{T}$.
- Therefore $\ell = \mathtt{T}$ is a forced literal for P.

Propagation and Learning

- At time t^*: $CurAsgn = \pi^*$, targetted subgame is f.
- Suppose $\pi^* \cup \{\neg\ell\}$ matches $\langle L_B^{\text{now}} \cup \{\neg\ell\}, L_B^{\text{fut}}\rangle \models (P \text{ loses } h)$, and
  - ℓ is owned by P.
  - ℓ does not appear outside h (and h is a subgame of f).
  - ℓ is upstream of all literals in L_B^{fut}. (ℓ gets picked before L_B^{fut})
- For P to win f, making $\ell = \texttt{F}$ is at least as bad as $\ell = \texttt{T}$.
- Therefore $\ell = \texttt{T}$ is a forced literal for P.
- Suppose $\pi^* \cup \{\ell\}$ matches $\underbrace{\langle L_A^{\text{now}} \cup \{\ell\},\ L_A^{\text{fut}}\rangle \models (P \text{ loses } f)}_{\text{in game-state database}}$.
- P loses f under $\pi^* \cup \{\ell\}$.
- P loses f under π^*, since $\ell = \texttt{F}$ is no better than $\ell = \texttt{T}$.

Propagation and Learning

- At time t^*: $CurAsgn = \pi^*$, targetted subgame is f.
- Suppose $\pi^* \cup \{\neg\ell\}$ matches $\langle L_B^{\text{now}} \cup \{\neg\ell\}, L_B^{\text{fut}}\rangle \models (P \text{ loses } h)$, and
  - ℓ is owned by P.
  - ℓ does not appear outside h (and h is a subgame of f).
  - ℓ is upstream of all literals in L_B^{fut}. (ℓ gets picked before L_B^{fut})
- For P to win f, making $\ell = \texttt{F}$ is at least as bad as $\ell = \texttt{T}$.
- Therefore $\ell = \texttt{T}$ is a forced literal for P.
- Suppose $\pi^* \cup \{\ell\}$ matches $\langle L_A^{\text{now}} \cup \{\ell\}, L_A^{\text{fut}}\rangle \models (P \text{ loses } f)$.
- Then learn: $\langle L_A^{\text{now}} \cup L_B^{\text{now}}, L_A^{\text{fut}} \cup L_B^{\text{fut}}\rangle \models (P \text{ loses } f)$.
  (Since the same argument applies to any matching assignment.)

Propagation and Learning

- At time t^*: $CurAsgn = \pi^*$, targetted subgame is f.
- Suppose $\pi^* \cup \{\neg\ell\}$ matches $\langle L_B^{\text{now}} \cup \{\neg\ell\}, L_B^{\text{fut}}\rangle \models (P \text{ loses } h)$, and
  - ℓ is owned by P.
  - ℓ does not appear outside h (and h is a subgame of f).
  - ℓ is upstream of all unassigned literals in L_B^{fut}.
- For P to win f, making $\ell = \mathtt{F}$ is at least as bad as $\ell = \mathtt{T}$.
- Therefore $\ell = \mathtt{T}$ is a forced literal for P.
- Suppose $\pi^* \cup \{\ell\}$ matches $\langle L_A^{\text{now}} \cup \{\ell\}, L_A^{\text{fut}}\rangle \models (P \text{ loses } f)$.
- Then learn: $\langle L_A^{\text{now}} \cup L_B^{\text{now}}, L_A^{\text{fut}} \cup L_B^{\text{fut}}\rangle \models (P \text{ loses } f)$.
  (Since the same argument applies to any matching assignment.)
- Move assigned literals from L_B^{fut} to L_B^{now} if upstream of ℓ.
  Then move back from $L_A^{\text{now}} \cup L_B^{\text{now}}$ to $L_A^{\text{fut}} \cup L_B^{\text{fut}}$.

Ghost Literals

- Goultiaeva et al. (SAT'09): propagation technique for circuit QBF.
  - Force a gate literal if detect that Player $\mathtt{E}$ needs it.
  - Asymmetric between players.
- We use *ghost literals* to make it symmetric:
  - Prenex: $g\langle\mathtt{U}\rangle$ for Player $\mathtt{U}$ and $g\langle\mathtt{E}\rangle$ for Player $\mathtt{E}$.
    - $g\langle P\rangle$ forced when detect P can win only if g is true.

Ghost Literals

- Goultiaeva et al. (SAT'09): propagation technique for circuit QBF.
  - Force a gate literal if detect that Player E needs it.
  - Asymmetric between players.
- We use *ghost literals* to make it symmetric:
  - Prenex: $g\langle \mathtt{U} \rangle$ for Player U and $g\langle \mathtt{E} \rangle$ for Player E.
  - Non-prenex: $g\langle \mathtt{U}, b \rangle$ and $g\langle \mathtt{E}, b \rangle$
    - b is a subgame which contains g
    - $g\langle P, b \rangle$ forced when detect P can win b only if g is true.
    - "Avoid a move that wins the battle but loses the war."

Optimized Ghost Literals

- Two tracks of QBFLIB benchmarks:
  1. CNF, reverse engr'd to prenex circuit form (DAG-based).
  2. Nonprenex NNF (tree-based representation of formula).
- Both tracks: No sharing of subformulas between subgames.
  - If a subformula directly occurs in two subgames, then the two occurrences are labelled with different gate vars.
- Optimization: See paper.

Experimental Results: GhostQ vs CirQit

- Implementation: GhostQ.
- Compare to CirQit (by Goultiaeva et al.) on QBFLIB non-CNF.

Disclosure:

- Different test machines. (CirQit not publicly available.)
- But CirQit had the advantage.
  GhostQ: 2.66 GHz, 300 sec
  CirQit: 2.80 GHz, 1200 sec

Family	inst.	GhostQ	CirQit
Seidl	*150*	**150**	147
assertion	*120*	**12**	3
consistency	*10*	0	0
counter	*45*	**40**	39
dme	*11*	**11**	10
possibility	*120*	**14**	10
ring	*20*	**18**	15
semaphore	*16*	16	16
Total	*492*	261	240

Experimental Results: GhostQ vs Qube

- QBFLIB CNF benchmarks.
- Timeout: 60 seconds.
- Reverse-engineer from CNF to circuit form.
- GhostQ beats Qube on `tipdiam`, `tipfixpoint`, `k`. (279 vs 173 solved instances.)

Family	inst.	GhostQ	Qube
bbox-01x	*450*	171	**341**
bbox_design	*28*	19	**28**
bmc	*132*	43	**49**
k	*61*	**42**	13
s	*10*	10	10
tipdiam	*85*	**72**	60
tipfixpoint	*196*	**165**	100
sort_net	*53*	0	**19**
all other	*121*	9	**23**
Total	*1136*	531	643

Conclusion

- Game-State Learning: Extend clause/cube learning.
- Ghost Literals: Symmetric propagation technique.
- Promising experimental results.
- Future work: Consider ghosting input variables for non-prenex? (Additional propagation power, but also more overhead.)