Circuit-based Search Space Pruning in QBF

Mikoláš Janota

IST/INESC-ID,
University of Lisbon, Portugal

Oxford, 10 July 2018, SAT

CQESTO

- Solver for quantified Boolean formulas
- In prenex, non-CNF form
- Operates directly on a circuit representation
- Uses SAT in a black-box fashion

CQESTO

- Solver for quantified Boolean formulas
- In prenex, non-CNF form
- Operates directly on a circuit representation
- Uses SAT in a black-box fashion

Remarks:

- Generalization of the CNF-based
  QESTO [Janota and Marques-Silva, 2015]
  CAQE [Rabe and Tentrup, 2015]
- Similar ideas implemented in Z3 for
  SMT [Bjørner and Janota, 2015]
  and QBF QuAbS [Tentrup, 2016]

Why Circuits?

- Known: CNF can be harmful for solving QBF [Ansótegui et al., 2005, Zhang, 2006, Janota and Marques-Silva, 2017]
- Intuition:
  We reason about formula and its negation.
  But, after Tseitin transformation,
  we do not have the negation!

QBF as Two-player Games

- A QBF is a game between $\forall$ and $\exists$

QBF as Two-player Games

- A QBF is a game between $\forall$ and $\exists$
- $\forall$ wins if the matrix becomes false

QBF as Two-player Games

- A QBF is a game between $\forall$ and $\exists$
- $\forall$ wins if the matrix becomes false
- $\exists$ wins if the matrix becomes true

QBF as Two-player Games

- A QBF is a game between $\forall$ and $\exists$
- $\forall$ wins if the matrix becomes false
- $\exists$ wins if the matrix becomes true
- QBF is false iff
  there exists a winning strategy for $\forall$

QBF as Two-player Games

- A QBF is a game between $\forall$ and $\exists$
- $\forall$ wins if the matrix becomes false
- $\exists$ wins if the matrix becomes true
- QBF is false iff
  there exists a winning strategy for $\forall$
- QBF is true iff
  there exists a winning strategy for $\exists$

QBF as Two-player Games

- A QBF is a game between $\forall$ and $\exists$
- $\forall$ wins if the matrix becomes false
- $\exists$ wins if the matrix becomes true
- QBF is false iff
  there exists a winning strategy for $\forall$
- QBF is true iff
  there exists a winning strategy for $\exists$

  Example

$$\forall u \exists e.\, (u \leftrightarrow e)$$

$\exists$-player wins by playing $e \triangleq u$

CQESTO: Architecture

$$
\begin{array}{ccccccc}
\boxed{\exists X_1} & \longrightarrow & \boxed{\forall X_2} & \cdots & \boxed{\forall X_{n-1}} & \longrightarrow & \boxed{\exists X_n} \\
\alpha_1 & & \alpha_2 & & \alpha_{n-1} & & \alpha_n
\end{array}
$$

- Propositional α_i for each level
- α_i restricts moves at position i
- Initially
$$
\begin{aligned}
\alpha_{n-1} &= \neg\mathsf{matrix} \\
\alpha_n &= \mathsf{matrix} \\
\alpha_i &= \mathsf{true}
\end{aligned}
$$

CQESTO: Algorithm

$$\begin{array}{ccccccccc}
\boxed{\exists X_1} & \to & \boxed{\forall X_2} & \cdots & \boxed{\exists X_i} & \cdots & \boxed{\forall X_{n-1}} & \to & \boxed{\exists X_n} \\
\alpha_1 & & \alpha_2 & & \alpha_i & & \alpha_{n-1} & & \alpha_n
\end{array}$$

- Assign values to X_i by calling SAT on α_i
- If α_i unsatisfiable, fix earlier mistake (strengthen previous $\alpha_{i'}$)
- If α_1 or α_2 unsatisfiable, the formula is proven (STOP)

CQESTO: Algorithm

$$\begin{array}{ccccccccc}
\sigma_1 & & & & & & & & \\
\boxed{\exists X_1} & \rightarrow & \boxed{\forall X_2} & \cdots & \boxed{\exists X_i} & \cdots & \boxed{\forall X_{n-1}} & \rightarrow & \boxed{\exists X_n} \\
\alpha_1 & & \alpha_2 & & \alpha_i & & \alpha_{n-1} & & \alpha_n
\end{array}$$

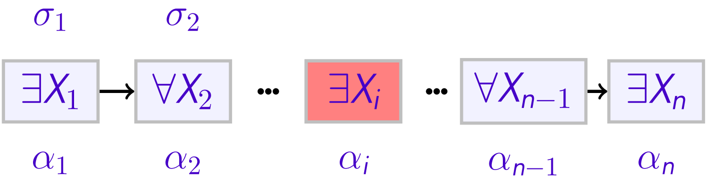
- Assign values to X_i by calling SAT on α_i
- If α_i unsatisfiable, fix earlier mistake (strengthen previous $\alpha_{i'}$)
- If α_1 or α_2 unsatisfiable, the formula is proven (STOP)

CQESTO: Algorithm

$$
\begin{array}{ccccccccc}
\sigma_1 & & \sigma_2 & & & & & & \\
\boxed{\exists X_1} & \rightarrow & \boxed{\forall X_2} & \cdots & \boxed{\exists X_i} & \cdots & \boxed{\forall X_{n-1}} & \rightarrow & \boxed{\exists X_n} \\
\alpha_1 & & \alpha_2 & & \alpha_i & & \alpha_{n-1} & & \alpha_n
\end{array}
$$

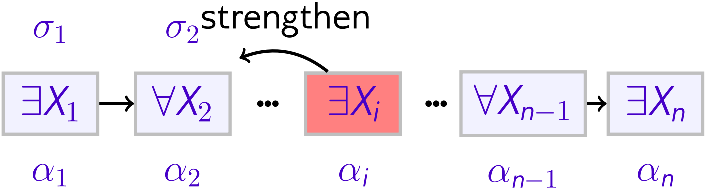
- Assign values to X_i by calling SAT on α_i
- If α_i unsatisfiable, fix earlier mistake (strengthen previous $\alpha_{i'}$)
- If α_1 or α_2 unsatisfiable, the formula is proven (STOP)

CQESTO: Algorithm



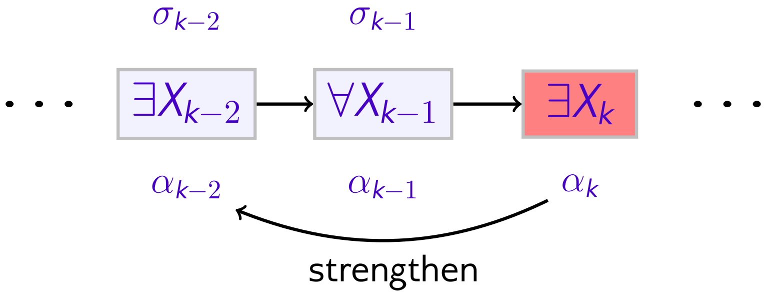
- Assign values to X_i by calling SAT on α_i
- If α_i unsatisfiable, fix earlier mistake (strengthen previous $\alpha_{i'}$)
- If α_1 or α_2 unsatisfiable, the formula is proven (STOP)

CQESTO: Algorithm



- Assign values to X_i by calling SAT on α_i
- If α_i unsatisfiable, fix earlier mistake (strengthen previous $\alpha_{i'}$)
- If α_1 or α_2 unsatisfiable, the formula is proven (STOP)

Loss Resolution (Conflicts)



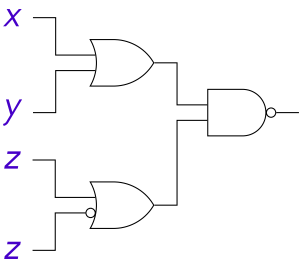
1. Identify reason R for failure in α_k.
2. Eliminate variables X_k from R
3. Eliminate variables X_{k-1} from R
4. Strengthen α_{k-2}

Reason (example)

$$(x \vee y) \to (z \wedge \neg z) = \neg((x \vee y) \wedge (z \vee \neg z))$$

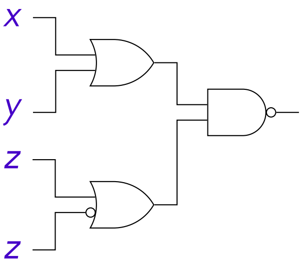
Reason (example)

$$(x \vee y) \rightarrow (z \wedge \neg z) = \neg((x \vee y) \wedge (z \vee \neg z))$$



Reason (example)

$$(x \vee y) \rightarrow (z \wedge \neg z) = \neg((x \vee y) \wedge (z \vee \neg z))$$



1. For $x = 1, y = 0$ propagate: $x \vee y = 1$
2. Give the gate a name α and use UNSAT cores to get the reason
3. Reason is $\alpha = x \vee y$

Eliminate X_{k-1}: plug in σ_{k-1}

1. $\exists x_1 x_2 \forall y \exists z.\ (x_1 \wedge z) \wedge (x_2 \vee y)$

Eliminate X_{k-1}: plug in σ_{k-1}

1. $\exists x_1 x_2 \forall y \exists z.\ (x_1 \wedge z) \wedge (x_2 \vee y)$
2. $\sigma_1 = \{x_1, \neg x_2\}$, $\sigma_2 = \{\neg y\}$.

Eliminate X_{k-1}: plug in σ_{k-1}

1. $\exists x_1 x_2 \forall y \exists z.\ (x_1 \wedge z) \wedge (x_2 \vee y)$
2. $\sigma_1 = \{x_1, \neg x_2\}$, $\sigma_2 = \{\neg y\}$.
3. propagation x_1 and $\neg(x_2 \vee y)$.

Eliminate X_{k-1}: plug in σ_{k-1}

1. $\exists x_1 x_2 \forall y \exists z.\ (x_1 \wedge z) \wedge (x_2 \vee y)$
2. $\sigma_1 = \{x_1, \neg x_2\}$, $\sigma_2 = \{\neg y\}$.
3. propagation x_1 and $\neg(x_2 \vee y)$.
4. core $\neg(x_2 \vee y)$.

Eliminate X_{k-1}: plug in σ_{k-1}

1. $\exists x_1 x_2 \forall y \exists z.\ (x_1 \wedge z) \wedge (x_2 \vee y)$
2. $\sigma_1 = \{x_1, \neg x_2\}$, $\sigma_2 = \{\neg y\}$.
3. propagation x_1 and $\neg(x_2 \vee y)$.
4. core $\neg(x_2 \vee y)$.
5. negating & substitute σ_2:
   $\xi_f = (x_2 \vee y)|_{\{\neg y\}} = x_2$

Eliminate X_{k-1}: plug in σ_{k-1}

1. $\exists x_1 x_2 \forall y \exists z.\ (x_1 \wedge z) \wedge (x_2 \vee y)$
2. $\sigma_1 = \{x_1, \neg x_2\}$, $\sigma_2 = \{\neg y\}$.
3. propagation x_1 and $\neg(x_2 \vee y)$.
4. core $\neg(x_2 \vee y)$.
5. negating & substitute σ_2:
   $\xi_f = (x_2 \vee y)|_{\{\neg y\}} = x_2$
6. strengthening $\alpha_1 \leftarrow \alpha_1 \wedge x_2$.

Eliminate X_k

Idea:

1. a formula $(x \wedge \phi) \vee (\neg x \wedge \psi)$

Eliminate X_k

Idea:

1. a formula $(x \wedge \phi) \vee (\neg x \wedge \psi)$
2. can be weakened by replacing x with 1

Eliminate X_k

Idea:

1. a formula $(x \wedge \phi) \vee (\neg x \wedge \psi)$
2. can be weakened by replacing x with 1
3. can be weakened by replacing $\neg x$ by 0

Eliminate X_k

Idea:

1. a formula $(x \land \phi) \lor (\neg x \land \psi)$
2. can be weakened by replacing x with 1
3. can be weakened by replacing $\neg x$ by 0
4. $\ldots \phi \lor \psi$

Eliminate X_k

Idea:

1. a formula $(x \wedge \phi) \vee (\neg x \wedge \psi)$
2. can be weakened by replacing x with 1
3. can be weakened by replacing $\neg x$ by 0
4. $\ldots \phi \vee \psi$

Eliminate X_k

Idea:

1. a formula $(x \wedge \phi) \vee (\neg x \wedge \psi)$
2. can be weakened by replacing x with 1
3. can be weakened by replacing $\neg x$ by 0
4. $\ldots \phi \vee \psi$

In general:

1. Replace positive occurrences by 1

Eliminate X_k

Idea:

1. a formula $(x \wedge \phi) \vee (\neg x \wedge \psi)$
2. can be weakened by replacing x with 1
3. can be weakened by replacing $\neg x$ by 0
4. $\ldots \phi \vee \psi$

In general:

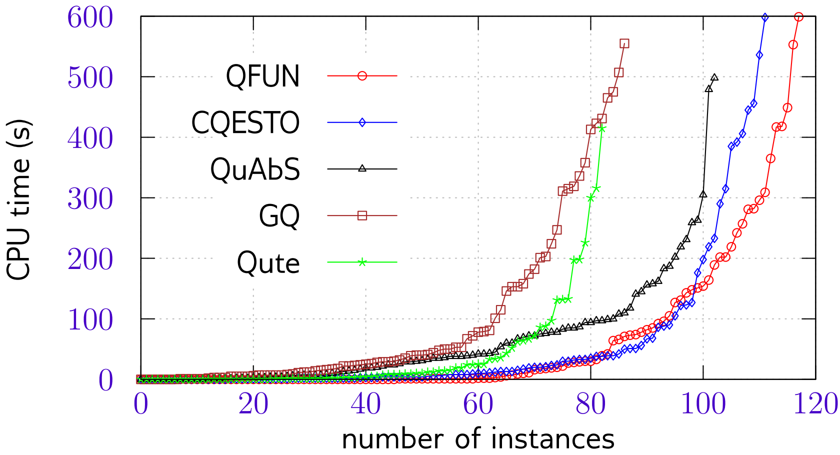
1. Replace positive occurrences by 1
2. Replace negative occurrences by 0

Loss Resolution (Conflicts): Recap

$$\begin{tikzcd}
 & \sigma_{k-2} & \sigma_{k-1} & & \\
\cdots & \boxed{\exists X_{k-2}} \arrow[r] & \boxed{\forall X_{k-1}} \arrow[r] & \boxed{\exists X_k} & \cdots \\
 & \alpha_{k-2} & \alpha_{k-1} & \alpha_k \arrow[ll, bend left, "\text{strengthen}"] &
\end{tikzcd}$$

1. Reason: propagation & SAT cores
2. Eliminate X_{k-1}: substitution of σ_{k-1}
3. Eliminate X_k: syntactic-based weakening
4. Strengthen α_{k-2} (or some $i \leq k-2$)

Experimental Evaluation



Summary and Future Work

- CQESTO works directly on circuit representation

Summary and Future Work

- CQESTO works directly on circuit representation
- Flat architecture (as opposed to RAReQS)

Summary and Future Work

- CQESTO works directly on circuit representation
- Flat architecture (as opposed to RAReQS)
- Conflict resolution by operations on circuit

Summary and Future Work

- CQESTO works directly on circuit representation
- Flat architecture (as opposed to RAReQS)
- Conflict resolution by operations on circuit
- Exact relation to other solvers?
  QELL [Tu et al., 2015] QuAbS [Tentrup, 2016]

Summary and Future Work

- CQESTO works directly on circuit representation
- Flat architecture (as opposed to RAReQS)
- Conflict resolution by operations on circuit
- Exact relation to other solvers?
  QELL [Tu et al., 2015] QuAbS [Tentrup, 2016]
- Identified: reason, eliminate opponent variables, eliminate player's variables

Summary and Future Work

- CQESTO works directly on circuit representation
- Flat architecture (as opposed to RAReQS)
- Conflict resolution by operations on circuit
- Exact relation to other solvers?
  QELL [Tu et al., 2015] QuAbS [Tentrup, 2016]
- Identified: reason, eliminate opponent variables, eliminate player's variables
- How are the identified processes realized in other solvers?

Summary and Future Work

- CQESTO works directly on circuit representation
- Flat architecture (as opposed to RAReQS)
- Conflict resolution by operations on circuit
- Exact relation to other solvers?
  QELL [Tu et al., 2015] QuAbS [Tentrup, 2016]
- Identified: reason, eliminate opponent variables, eliminate player's variables
- How are the identified processes realized in other solvers?
- Different ways of realizing processes?

- Ansótegui, C., Gomes, C. P., and Selman, B. (2005).
  The Achilles' heel of QBF.
  In *National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference (AAAI)*, pages 275–281.

- Bjørner, N. and Janota, M. (2015).
  Playing with quantified satisfaction.
  In *International Conferences on Logic for Programming LPAR-20, Short Presentations*, volume 35, pages 15–27. EasyChair.

- Janota, M. and Marques-Silva, J. (2015).
  Solving QBF by clause selection.

In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Janota, M. and Marques-Silva, J. (2017).
An Achilles' heel of term-resolution.
In *Conference on Artificial Intelligence (EPIA)*, pages 670–680.

Rabe, M. N. and Tentrup, L. (2015).
CAQE: A certifying QBF solver.
In *Formal Methods in Computer-Aided Design, FMCAD*, pages 136–143.

Tentrup, L. (2016).
Non-prenex QBF solving using abstraction.

In *Theory and Applications of Satisfiability Testing (SAT)*, pages 393–401.

Tu, K., Hsu, T., and Jiang, J. R. (2015).
QELL: QBF reasoning with extended clause learning and levelized SAT solving.
In *Theory and Applications of Satisfiability Testing - (SAT)*, pages 343–359.

Zhang, L. (2006).
Solving QBF by combining conjunctive and disjunctive normal forms.
In *National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference (AAAI)*.