Efficient Extraction of Skolem Functions from QRAT Proofs

Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

Joint work with

Martina Seidl and Armin Biere

JOHANNES KEPLER UNIVERSITY LINZ | JKU

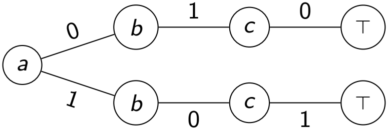
FMCAD, October 23, 2014

Introduction to QBF

A quantified Boolean formula (QBF) is a propositional formula where variables are existentially ($\exists$) or universally ($\forall$) quantified.

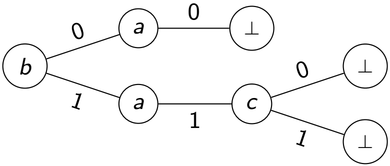
Consider the formula $\forall a\, \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

A model is:



Consider the formula $\exists b\, \forall a\, \exists c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

A counter-model is:

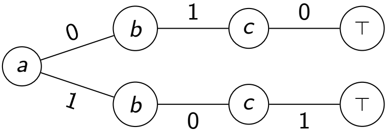


Introduction to Skolem functions for QBF

A Skolem function $f_x(U_x)$ for a QBF formula $\pi.\psi$ defines the truth value of an existential variable x based on the set U_x of universal variables that occur earlier in the prefix than x

Consider the formula $\forall a\, \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

A model is:



The set of Skolem functions F (defining all existentials) is

$$F = \{f_b(a) = \neg a,\ f_c(a) = a\}$$

The set of Skolem functions can be much smaller than a model

Challenges for Quantified Boolean Formulas (QBF)

Preprocessing is crucial to solve most QBF instances efficiently.

Proofs are useful for applications and to validate solver output.

Main challenges regarding QBF and preprocessing [Janota'13]:

1. produce proofs that can be validated in polynomial time;
2. develop methods to validate all QBF preprocessing; and
3. narrow the performance gap between solving with and without proof generation.

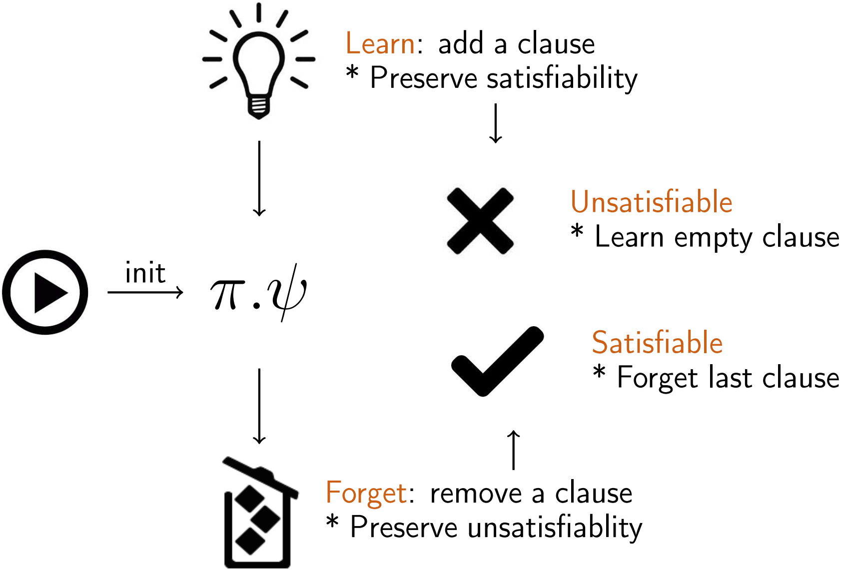
In our IJCAR'14 paper [1], we meet all three challenges!

[1] Marijn J. H. Heule, Matina Seidl and Armin Biere:
A Unified Proof System for QBF Preprocessing.
IJCAR 2014, LNCS 8562, pp 91-106 (2014)

Here we show how to make Skolem functions out of the proofs.

From Clausal Proofs to Skolem Functions

Clausal Proof System



Redundancy Concepts in the QRAT Proof System

Informal definitions of the redundancy concepts in the QRAT proof system. They can be computed in polynomial time.

Definition (Asymmetric Tautologies (AT))

An asymmetric tautology is a clause that becomes a tautology after adding "hidden literals". ATs are logically implied by a formula.

Definition (Quantified Resolution AT (QRAT))

A quantified resolution AT is a clause that contains a literal for which all "outer resolvents" are ATs.

Definition (Extended Universal Reduction (EUR))

A universal literal is redundant if assigning it to false cannot influence the value of universal literals.

Rules of the QRAT Proof System

	Rule	Preconditions	Postconditions
(N1)	$\dfrac{\pi.\psi}{\pi.\psi \backslash \{C\}}$	C is an asymmetric tautology	
(N2)	$\dfrac{\pi.\psi}{\pi'.\psi \cup \{C\}}$	C is an asymmetric tautology	$\pi' = \pi\exists X$ with $X = \{x \mid x \in \text{vars}(C), x \notin \text{vars}(\pi)\}$
(E1)	$\dfrac{\pi.\psi}{\pi.\psi \backslash \{C\}}$	$C \in \psi$, $\text{Q}(\pi, l) = \exists$ C has QRAT on l w.r.t. ψ	
(E2)	$\dfrac{\pi.\psi}{\pi'.\psi \cup \{C\}}$	$C \notin \psi$, $\text{Q}(\pi, l) = \exists$ C has QRAT on l w.r.t. ψ	$\pi' = \pi\exists X$ with $X = \{x \mid x \in \text{vars}(C), x \notin \text{vars}(\pi)\}$
(U1)	$\dfrac{\pi.\psi \cup \{C\}}{\pi.\psi \cup \{C \backslash \{l\}\}}$	$l \in C$, $\text{Q}(\pi, l) = \forall$, $\neg l \notin C$, C has QRAT on l w.r.t. ψ	
(U2)	$\dfrac{\pi.\psi \cup \{C\}}{\pi.\psi \cup \{C \backslash \{l\}\}}$	$l \in C$, $\text{Q}(\pi, l) = \forall$, $\neg l \notin C$, C has EUR on l w.r.t. ψ	

Rules of the QRAT Proof System

	Rule	Preconditions	Postconditions
(N1)	$\dfrac{\pi.\psi}{\pi.\psi\backslash\{C\}}$	C i … tau …	Preserves Logical Equivalence
(N2)	$\dfrac{\pi.\psi}{\pi'.\psi\cup\{C\}}$	C i … tau …	Preserves Logical Equivalence … $(\pi)\}$
(E1)	$\dfrac{\pi.\psi}{\pi.\psi\backslash\{C\}}$	$C \in \psi$, Q(… C has QR…	Weakens the Formula
(E2)	$\dfrac{\pi.\psi}{\pi'.\psi\cup\{C\}}$	$C \notin \psi$, C has (…	Strengthens the Formula … vars$(\pi)\}$
(U1)	$\dfrac{\pi.\psi\cup\{C\}}{\pi.\psi\cup\{C\backslash\{l\}\}}$	$l \in C$, Q … C has (…	Strengthens the Formula
(U2)	$\dfrac{\pi.\psi\cup\{C\}}{\pi.\psi\cup\{C\backslash\{l\}\}}$	$l \in C$, Q … C has E …	Strengthens the Formula

Pseudo-Code of Skolem Function Computation

ComputeSkolem (prefix π, QRAT proof P)

let ψ be an empty formula
foreach existential variable e **do** $f_e(U) := *$ // initialize F
while (P is not empty) **do**
  $\langle \textit{rule } R, \textit{clause } C, \textit{literal } l \rangle := P.\mathrm{pop}()$
  if (R = E1) **then**
    let e be var(l)
    $f_e(U) := \mathrm{IfThenElse}(F(\mathcal{OF}(\pi, \psi, l)), \mathrm{polarity}(l), f_e(U))$
  if (R = E1 **or** R = N1) **then** // Forget rules
    $\psi := \psi \cup \{C\}$
  if (R = E2 **or** R = N2) **then** // Learn rules
    $\psi := \psi \setminus \{C\}$

Adding a Skolem Function

The outer clause of D w.r.t. a literal l under prefix π is:

$$\mathcal{OC}(\pi, D, l) := \{k \mid k \in D, \pi(k) \leq \pi(l), \text{ and } k \neq l\}$$

The outer formula of ψ w.r.t. a literal l under prefix π is:

$$\mathcal{OF}(\pi, \psi, l) := \{\mathcal{OC}(\pi, D, \neg l) \mid D \in \psi, \neg l \in D\}$$

How to understand

$$f_e(U) := \mathrm{IfThenElse}(F(\mathcal{OF}(\pi, \psi, l))), \mathsf{polarity}(l), f_e(U)) \ ?$$

If a clause C has QRAT on literal $l \in C$ w.r.t. ψ, then

- any assignment that falsifies $\mathcal{OF}(\pi, \psi, l)$ satisfies C
- if $\mathcal{OF}(\pi, \psi, l)$ is satisfied, we can safely assign l to true

Running Example

Running Example

Consider again $\pi.\psi := \forall a \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

QRAT proof P using the rules E1 (Forget) and E2 (Learn):

$$\text{E2}(\neg a \vee \neg b), \text{E1}(\neg a \vee c), \text{E1}(\neg b \vee \neg c), \text{E1}(\neg a \vee \neg b), \text{E1}(a \vee b)$$

Rule	ψ	$\mathcal{OF}(\pi, \psi, l)$	**Skolem set** F
init	$\emptyset$	n\a	$f_b(a) = *,\ f_c(a) = *$

Running Example

Consider again $\pi.\psi := \forall a \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

QRAT proof P using the rules E1 (Forget) and E2 (Learn):

$E2(\neg a \vee \neg b), E1(\neg a \vee c), E1(\neg b \vee \neg c), E1(\neg a \vee \neg b), E1(a \vee b)$

Rule	ψ	$\mathcal{OF}(\pi, \psi, I)$	**Skolem set** F
init	$\emptyset$	n\a	$f_b(a) = *,\ f_c(a) = *$
$E1(a \vee b)$	$\emptyset$	$\emptyset$	$f_b(a) = \top,\ f_c(a) = *$

Running Example

Consider again $\pi.\psi := \forall a\, \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

QRAT proof P using the rules E1 (Forget) and E2 (Learn):

$$E2(\neg a \vee \neg b), E1(\neg a \vee c), E1(\neg b \vee \neg c), E1(\neg a \vee \neg b), E1(a \vee b)$$

Rule	ψ	$\mathcal{OF}(\pi, \psi, l)$	**Skolem set** F
init	$\emptyset$	n\a	$f_b(a) = *,\ f_c(a) = *$
E1$(a \vee b)$	$\emptyset$	$\emptyset$	$f_b(a) = \top,\ f_c(a) = *$
E1$(\neg a \vee \neg b)$	$(a \vee b)$	(a)	$f_b(a) = \neg a,\ f_c(a) = *$

Running Example

Consider again $\pi.\psi := \forall a \, \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

QRAT proof P using the rules E1 (Forget) and E2 (Learn):

$$\mathrm{E2}(\neg a \vee \neg b), \mathrm{E1}(\neg a \vee c), \mathrm{E1}(\neg b \vee \neg c), \mathrm{E1}(\neg a \vee \neg b), \mathrm{E1}(a \vee b)$$

Rule	ψ	$\mathcal{OF}(\pi, \psi, I)$	**Skolem set** F
init	$\emptyset$	n\a	$f_b(a) = *,\ f_c(a) = *$
$\mathrm{E1}(a \vee b)$	$\emptyset$	$\emptyset$	$f_b(a) = \top,\ f_c(a) = *$
$\mathrm{E1}(\neg a \vee \neg b)$	$(a \vee b)$	(a)	$f_b(a) = \neg a,\ f_c(a) = *$
$\mathrm{E1}(\neg b \vee \neg c)$	$(a \vee b) \wedge (\neg a \vee \neg b)$	$\emptyset$	$f_b(a) = \neg a,\ f_c(a) = \bot$

Running Example

Consider again $\pi.\psi := \forall a \exists b, c.(a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c)$

QRAT proof P using the rules E1 (Forget) and E2 (Learn):

$$\text{E2}(\neg a \vee \neg b), \text{E1}(\neg a \vee c), \text{E1}(\neg b \vee \neg c), \text{E1}(\neg a \vee \neg b), \text{E1}(a \vee b)$$

Rule	ψ	$\mathcal{OF}(\pi, \psi, l)$	**Skolem set** F
init	$\emptyset$	n\a	$f_b(a) = *,\ f_c(a) = *$
$\text{E1}(a \vee b)$	$\emptyset$	$\emptyset$	$f_b(a) = \top,\ f_c(a) = *$
$\text{E1}(\neg a \vee \neg b)$	$(a \vee b)$	(a)	$f_b(a) = \neg a,\ f_c(a) = *$
$\text{E1}(\neg b \vee \neg c)$	$(a \vee b) \wedge (\neg a \vee \neg b)$	$\emptyset$	$f_b(a) = \neg a,\ f_c(a) = \bot$
$\text{E1}(\neg a \vee c)$	$(a \vee b) \wedge (\neg a \vee \neg b) \wedge (\neg b \vee \neg c)$	$(\neg b)$	$f_b(a) = \neg a,\ f_c(a) = \neg f_b(a)$

Validating Skolem Functions

Checks to Validate Skolem Functions

Two tests are required to validate Skolem functions:

1. Can we falsify a clause in formula ψ while satisfying the Skolem functions $F(U)$?

   $$\text{solve}(\neg\psi \wedge F(U)) \ = \text{UNSAT?}$$

2. Check that all Skolem functions depend only on universal variables that occur earlier in the prefix.

   Problem: our method could create a Skolem function

   $$f_x(U_x) := f_y(U_y) \text{ with } \pi(x) < \pi(y)$$

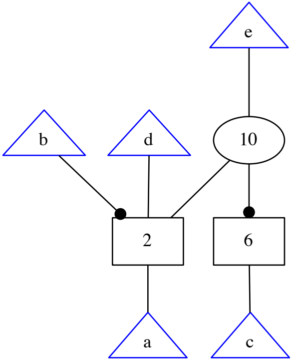
   Solution: convert Skolem functions to And-Inverter-Graphs (AIGs) and check for reachability.

Check Reachability in AIGs

Consider the formula $\pi.\psi$:

$$
\begin{aligned}
\forall a \exists b \forall c \exists d, e.& \\
(a \vee b) \quad &\wedge \\
(\neg a \vee \neg b \vee d) \quad &\wedge \\
(a \vee c \vee \neg d) \quad &\wedge \\
(a \vee \neg b \vee \neg e) \quad &\wedge \\
(\neg a \vee c \vee e) \quad &\wedge \\
(\neg c \vee \neg e) &
\end{aligned}
$$

Skolem functions for $\pi.\psi$:



Our algorithm could have produced $f_b(a) := f_d(a, c)$, but that is not problematic because $f_d(a, c)$ does not depend on c.

How to simplify the circuit and preserve the dependencies?

Experimental Results

Experimental Results: Solving versus Extraction

We used the benchmarks of QBF Eval 2012 as the test set.

First, we compare the costs of solving true QBF formulas and the costs to extract Skolem functions from the proofs

- Extraction of Skolem functions includes proof validation

Summary of the results of the first experiment:

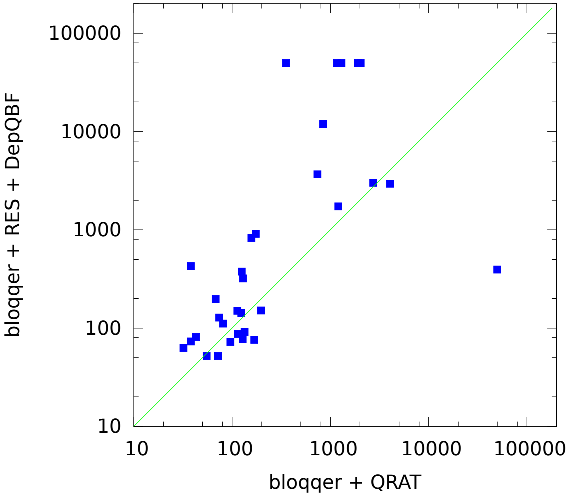
- Extraction costs of Skolem functions is comparable to solving time. The theoretical worst-case is polynomial.
- The size of the set of Skolem functions is linear in the solving time: a few megabyte (AAG format) per second.
- Validating the Skolem functions is comparable to the extraction time, but can be an order of magnitude slower.

Experimental Results: Comparison with other Tools

solver	sol-#	sol-t	ch-#	ch-t	cer-s
bloqqer+QRAT	**32**	1	**32**	47	1851
bloqqer+RES	22	1	22	1	861
bloqqer+RES+depQBF	28	113	27	13	1040
depQBF	2	843	2	1	224
ebdd	15	491	7	118	409479
squolem	16	465	16	2	382
sKizzo	23	275	23	1	108750

sol-#: # solved formulas, sol-t: avg. solving time (s),
ch-#: checked certificates, ch-t: avg. checking time (s)
cer-s: avg. certificate size (kilobyte)

Experimental Results: Size Comparison



Above the diagonal: Skolem functions from QRAT proofs are smaller

Conclusions

Conclusions

Compute Skolem functions out of QRAT proofs:

- All QBF preprocessing techniques can be stated in QRAT
- The proof size is polynomial in solving time (worst-case)
- We showed how to convert QRAT into Skolem functions
- The size of Skolem functions is relatively small: Linear in the size of proofs in practice, polynomial in worst-case

Directions for future work:

- How to state all QBF solving techniques in QRAT?
  - That would allow Skolem functions for the full QBF tool chain
- Shrink Skolem functions using circuit simplification
  - There are strong circuit simplification tools around, e.g. ABC

Conclusions

Compute Skolem functions out of QRAT proofs:

- All QBF preprocessing techniques can be stated in QRAT
- The proof size is polynomial in solving time (worst-case)
- We showed how to convert QRAT into Skolem functions
- The size of Skolem functions is relatively small: Linear in the size of proofs in practice, polynomial in worst-case

Directions for future work:

- How to state all QBF solving techniques in QRAT?
  - That would allow Skolem functions for the full QBF tool chain
- Shrink Skolem functions using circuit simplification
  - There are strong circuit simplification tools around, e.g. ABC

Thanks!