Cost Minimizing Search on Non-prenex and Non-CNF QBF[1]

Soonho Kong

Programming Research Laboratory
Seoul National University

March 21, 2008

[1] Work done with Dr. Ting Zhang in Microsoft Research Asia

QBF

Quantified Boolean Formulae

- Quantifiers: $\forall, \exists$
- All variables are Boolean variables.
- Logical Operator: $\wedge, \vee, \neg$

Example

$$\psi_1 = \forall x \exists y (x \vee \neg y)$$

True

Example

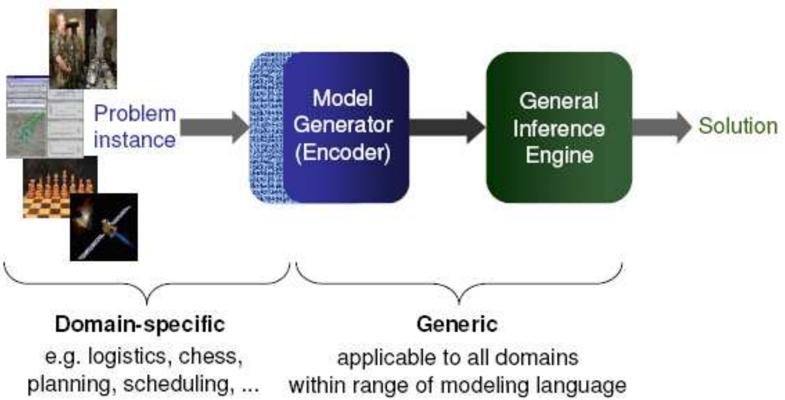
$$\psi_2 = \forall x \exists y (x \wedge \neg y)$$

False

QBF Problem

- The QBF problem is to determine its truth value.
- In complexity theory, it is PSPACE problem.
- At least as hard as NP.

Why is QBF solving important?



[Credit: A. Sabharwal, B. Selman, Beyond Traditional SAT Reasoning, AAAI 2007 Tutorial]

Why is QBF solving important?

QBF compactly encode many problems in CSE.

- Automated Plannig
- Scheduling
- Bounded Model Checking(BMC)
- Electronic Design Automation

Improvement in solving engine affects many application areas.

Compact Encoding

- QBF provides a compacter encoding than SAT.
- Transforming QBF into equivalent SAT increases space exponentially.
  - QBF : $\forall x \exists y \exists z (x \vee \neg y \vee z)$
  - SAT : $(\exists y \exists z (\top \vee \neg y \vee z)) \wedge (\exists y \exists z (\bot \vee \neg y \vee z))$

Existing Approach

- Search-based
- Q-Resolution
- Symbolic Skolemization

Existing Approach

Search-based QBF solving is dominant.

- Search-based technique is dominant in SAT solving.
- People tried to apply and extend those techniques to QBF solving problem.

Search-based QBF solving

Search-based QBF solvers require normalized input.

- Prenex Form
- Conjunctive Normal Form(CNF)

Normalized Input - Prenex Form

Prenex Form

- (X) : $\forall x(\exists y(x \vee y) \wedge (\exists z(x \vee z)))$
- (O) : $\forall x \exists y \exists z((x \vee y) \wedge (x \vee z))$

Normalized Input - CNF

Conjunctive Normal Form(CNF)

- Literal : Variable or Negation of Variable
  $x, \neg x$
- Clause : Disjucntion of Literals
  $(x \vee \neg y \vee z)$
- CNF : Conjunction of Clauses
  $(x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$

Why should be “prenex and CNF”?

Binary Constraint Propagation(BCP)

Why “prenex and CNF”?

Binary Constraint Propagation(BCP)

BCP propagate one assignment
and deduce another another assignment
without searching the space.

Why "prenex and CNF"?

BCP is a "Founding block" for

- non-chronological backtracking
- conflict-driven learning

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\forall x \exists y \exists z ((x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z))$$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\forall x \exists y \exists z ((x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z))$$

Try $x = \top$.

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists y \exists z((\top \vee \neg y \vee z) \wedge (\bot \vee \neg y \vee \neg z))$$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists y \exists z (\neg y \vee \neg z)$$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists y \exists z (\neg y \vee \neg z)$$

Try $y = \top$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists z(\perp \vee \neg z)$$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists z(\neg z)$$

Now, z must be $\perp$ and it makes the formula satisfied.

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\forall x \exists y \exists z ((x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z))$$

Go back to try $x = \perp$ because of $\forall x$.

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\forall x \exists y \exists z ((\bot \lor \neg y \lor z) \land (\top \lor \neg y \lor \neg z))$$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists y \exists z (\neg y \vee z)$$

Try $y = \top$.

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists z(\perp \vee z)$$

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\exists z(\vee z)$$

Now we know that z should be $\top$
and it makes the formula satisfied.

Binary Constraint Propagation(BCP) Example

Binary Constraint Propagation(BCP)

$$\forall x \exists y \exists z ((\bot \vee \neg y \vee z) \wedge (\top \vee \neg y \vee \neg z))$$

Now we know that the formula is true.

Problem

It is necessary to have prenex and CNF to conduct BCP.

Problem

However,
QBF encodings from the real world applications
never come in prenex and CNF.

Problem

Prenex converting and CNF converting are not free.

Problem - CNF

Converting into equivalent CNF takes exponential time.

Problem - CNF

Tseytin introduced linear-time CNF converting method.
However,
it introduces new existentially quantified variables.

Problem - CNF

Our experiment shows
total number of variables increase by
four times.

Problem - prenex form

Converting into prenex form enlarges the search space by

- widening quantifier scope
- introducing dependency to the variables which were independent

Problem - prenex form

$$\forall x \psi_1(x) \wedge \exists y \psi_2(y) \wedge \forall z \psi_3(z) \wedge \exists w \psi_4(w)$$

is converted into

$$\forall x \exists y \forall z \exists w (\psi_1(x) \wedge \psi_2(y) \wedge \psi_3(z) \wedge \psi_4(w))$$

Motivation

Is there another way to solve QBF
without converting input into prenex and CNF?

Motivation

Is there another direction in QBF solving
which is not based on boolean constraint propagation?

Motivation

Yes, there is!

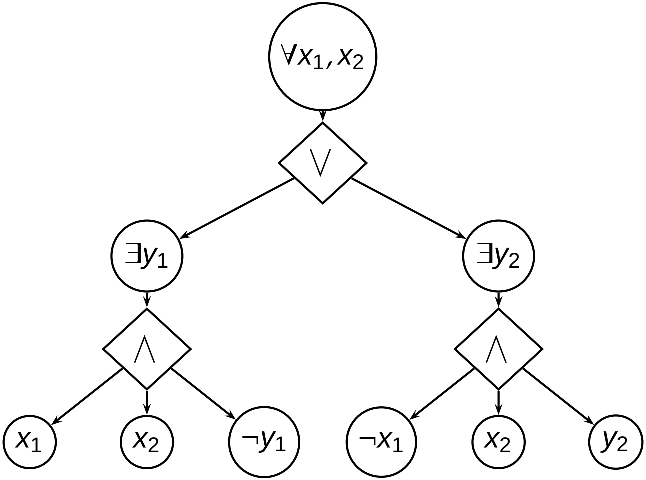
Naive algorithm solves QBF without BCP.

Motivation - Naive Algorithm

$$
\begin{array}{lll}
[\![\top]\!] := \top & [\![\neg\top]\!] := \bot & [\![\bigwedge_i \varphi_i]\!] := \bigwedge_i [\![\varphi_i]\!] \\
[\![\bot]\!] := \bot & [\![\neg\bot]\!] := \top & [\![\bigvee_i \varphi_i]\!] := \bigvee_i [\![\varphi_i]\!]
\end{array}
$$

$$
\begin{aligned}
[\![\forall x\ \varphi]\!] &:= [\![\varphi[x \mapsto \top]]\!] \ \wedge\ [\![\varphi[x \mapsto \bot]]\!] \\
[\![\forall \bar{x}\ \varphi]\!] &:= [\![\forall(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \top])]\!] \ \wedge\ [\![\forall(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \bot])]\!] \\
[\![\exists x\ \varphi]\!] &:= [\![\varphi[x \mapsto \top]]\!] \ \vee\ [\![\varphi[x \mapsto \bot]]\!] \\
[\![\exists \bar{x}\ \varphi]\!] &:= [\![\exists(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \top])]\!] \ \vee\ [\![\exists(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \bot])]\!]
\end{aligned}
$$

Example



Motivation - Naive Algorithm

But its performance is poor.
2 - 4 orders of magnitude slower
than the state-of-the-art solvers.

Motivation

Is there an "efficient" search-based method to solve QBF without conducting boolean constraint propagation?

Cost Minimizing Search

Cost Minimizing Search
Based on the same naive algorithm shown before.

Preprocessing

Push Quantifiers as Further as Possible
to Reduce Search Space

$$\forall x(\varphi \wedge \psi) \equiv (\forall x \varphi) \ \wedge \ \psi, \qquad \forall x(\varphi \vee \psi) \equiv (\forall x \varphi) \ \vee \ \psi,$$
$$\exists x(\varphi \wedge \psi) \equiv (\exists x \varphi) \ \wedge \ \psi, \qquad \exists x(\varphi \vee \psi) \equiv (\exists x \varphi) \ \vee \ \psi.$$

$\varphi, \psi \in$ QBF and x a variable not occurring *free* in ψ. For example

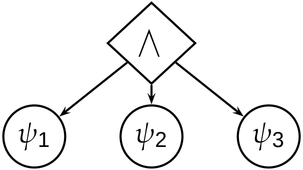
$$\forall x \exists y \exists z((x \vee y) \wedge (x \vee z)) \rightarrow \forall x(\exists y(x \vee y) \wedge (\exists z(x \vee z)))$$

Observation

We can reduce the time
by exploiting "short-circuit" property of $\bigwedge$ and $\bigvee$ operator.

Observation

Once we found a desired result from the subnode,
We do not have to continue the evaluation.



If ψ_1 is false, then whole formula turns out to be false.

Observation

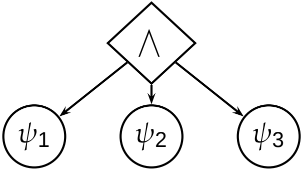
Since operators $\bigwedge$ and $\bigvee$ are *commutative*,
we can *sort* the operands and exploit short-circuit property.

Considerations

Consider two things:

1. Truth Probability
2. Expected Search Space

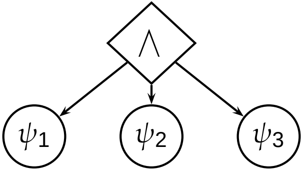
Truth Probability



$$P(\psi_1) = 0.8, P(\psi_2) = 0.6, P(\psi_3) = 0.3$$

It is better to sort ψ_i to ψ_3, ψ_2, ψ_1 and try ψ_3 first.

Expected Search Space



$$S(\psi_1) = 300, S(\psi_2) = 60, S(\psi_3) = 90$$

It is better to sort ψ_i to ψ_2, ψ_3, ψ_1 and try ψ_2 first.

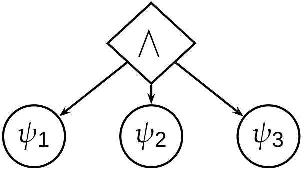
Cost

In general, truth probability and expected search space vary at the same time. We need a *single measure* to sort the subnode ψ_is. We introduce the concept of *cost*.

$$\text{Cost} = \text{Probability of Being Useless} \times \text{Expected Search Space}$$

Cost Minimizing Search

Conduct depth-first cost minimizing search



$$
\begin{array}{lll}
P(\psi_1) = 0.8 & P(\psi_2) = 0.6 & P(\psi_3) = 0.3 \\
S(\psi_1) = 300 & S(\psi_2) = 60 & S(\psi_3) = 90 \\
C(\psi_1) = 24 & C(\psi_2) = 36 & C(\psi_3) = 27
\end{array}
$$

It is better to sort ψ_i to ψ_1, ψ_3, ψ_2 and try ψ_1 first.

Probability Calculation

1. $\varphi \equiv x \mid \neg x$

$$\mathrm{P}(x) = \mathrm{P}(\neg x) = \frac{1}{2}$$

2. $\varphi \equiv \bigwedge_{i=1}^{n} \varphi_i$

$$\mathrm{P}\left(\bigwedge_{i=1}^{n} \varphi_i\right) = \prod_{i=1}^{n} \mathrm{P}(\varphi_i)$$

3. $\varphi \equiv \bigvee_{i=1}^{n} \varphi_i$

$$\mathrm{P}\left(\bigvee_{i=1}^{n} \varphi_i\right) = 1 - \prod_{i=1}^{n} (1 - \mathrm{P}(\varphi_i))$$

Probability Calculation

4. $\varphi \equiv \forall \bar{x} \psi$

$$\mathrm{P}(\forall \bar{x} \psi) = \prod_{i=1}^{2^{|\bar{x}|}} \mathrm{P}(\psi) = (\mathrm{P}(\psi))^{(2^{|\bar{x}|})}$$

5. $\varphi \equiv \exists \bar{x} \psi$

$$\mathrm{P}(\neg \forall \bar{x} \neg \psi) = 1 - \mathrm{P}(\forall \bar{x} \neg \psi) = 1 - (1 - \mathrm{P}(\psi))^{(2^{|\bar{x}|})}$$

Expected Search Space Calculation

1. $\varphi \equiv x \mid \neg x$

$$S(x) = S(\neg x) = 1$$

Expected Search Space Calculation

2 $\varphi \equiv \bigwedge_{i=1}^{n} \varphi_i$

$$\mathrm{S}\left(\bigwedge_{i=1}^{n} \varphi_i\right) = \sum_{i=1}^{n} \widehat{\mathrm{P}_i}(\varphi)\widehat{\mathrm{S}_i}(\varphi) + \left(\prod_{j=1}^{n} \mathrm{P}(\varphi_j)\right)\left(\sum_{j=1}^{n} \mathrm{S}(\varphi_j)\right)$$

$$\widehat{\mathrm{P}_i}(\varphi) = \left(\prod_{j=1}^{i-1} \mathrm{P}(\varphi_j)\right)(1 - \mathrm{P}(\varphi_i)),$$

$$\widehat{\mathrm{S}_i}(\varphi) = \sum_{j=1}^{i} \mathrm{S}(\varphi_j)$$

where $\widehat{\mathrm{P}_i}(\varphi)$ is the probability that $[\![\varphi_j]\!] = \top$ for any $1 \leq j \leq i-1$ and $[\![\varphi_i]\!] = \bot$.

Expected Search Space Calculation

2. $\varphi \equiv \bigwedge_{i=1}^{n} \varphi_i$ Introducing an imaginary element φ_{n+1} such that $\mathrm{P}(\varphi_{n+1}) = 0$ and $\mathrm{S}(\varphi_{n+1}) = 0$, we simplify to

$$\mathrm{S}\left(\bigwedge_{i=1}^{n} \varphi_i\right) = \sum_{i=1}^{n+1} \widehat{\mathrm{P}}_i(\varphi)\widehat{\mathrm{S}}_i(\varphi)$$

Expected Search Space Calculation

3. $\varphi \equiv \bigvee_{i=1}^{n} \varphi_i$

$$S\left(\bigvee_{i=1}^{n} \varphi_i\right) = \sum_{i=1}^{n+1} \widetilde{P}_i(\varphi)\widehat{S}_i(\varphi)$$

$$\widetilde{P}_i(\varphi) = \left(\prod_{j=1}^{i-1} \Big(1 - P(\varphi_j)\Big)\right) P(\varphi_i)$$

   where $P(\varphi_{n+1})$ is an imaginary element set to 1.

4. $\varphi \equiv \forall \bar{x} \psi$ In the general case where $\bar{x}$ has $|\bar{x}|$ elements, we have

$$S(\forall \bar{x} \psi) = (1 + P(\psi))^{|\bar{x}|} S(\psi)$$

5. $\varphi \equiv \exists \bar{x} \psi$

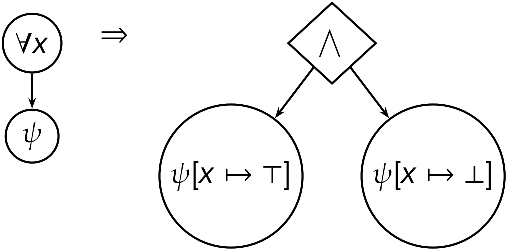
$$S(\exists \bar{x} \psi) = (2 - P(\psi))^{|\bar{x}|} S(\psi)$$

Cost Calculation

1. $\varphi \equiv x \mid \neg x$
   Leaf nodes have no child and hence no cost assigning is needed.
2. $\varphi \equiv \bigwedge_{i=1}^{n} \varphi_i$
   $$C(\varphi_i) = P(\varphi_i)S(\varphi_i)$$
3. $\varphi \equiv \bigvee_{i=1}^{n} \varphi_i$
   $$C(\varphi_i) = (1 - P(\varphi_i))S(\varphi_i)$$
4. $\varphi \equiv \forall \bar{x} \psi$
   $$C(\psi_i) = P(\psi_i)S(\psi_i)$$
5. $\varphi \equiv \exists \bar{x} \psi$
   $$C(\psi_i) = (1 - P(\psi_i))S(\psi_i)$$

Quantifier Node

During the evaluation quantifier will be replaced
with logical operator $\bigwedge$(or $\bigvee$) and instantiated child node.



Quantifier Node

Therefore, we can apply CMS for the quantifier.

Instead of sorting the subnodes,
we control the variable instantiation order.

Variable Instantiation Order

If a quantifier node has n variables,
then there could be 2^n variable instantiations.

$$\forall\{x_1, x_2, \ldots, x_n\}\psi$$

$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \top, \ldots, x_n \mapsto \top] = ?$$
$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \top, \ldots, x_n \mapsto \bot] = ?$$
$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \bot, \ldots, x_n \mapsto \top] = ?$$
$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \bot, \ldots, x_n \mapsto \bot] = ?$$
$$\ldots$$

Variable Instantiation Order

Naive approach would take 2^n cost calculations
to find minimal cost variable instantiation.

That is prohibitively expensive.

Variable Instantiation Order

We approximate the cost of an instantiation

$$\mathrm{C}(\psi[x_1 \mapsto b_1][x_2 \mapsto b_2]\cdots[x_n \mapsto b_n]) = \frac{\sum_{i=1}^{n} \mathrm{C}(\psi[x_i \mapsto b_i])}{n}$$

Using this approximation
we only need to compute $2n$ single-variable instantiations.

Variable Instantiation Order

However, it is not efficient enough
as we still have $2^{|\bar{x}|}$ combinations to compute and sort.

$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \top, \ldots, x_n \mapsto \top]) = ?$$
$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \top, \ldots, x_n \mapsto \bot]) = ?$$
$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \bot, \ldots, x_n \mapsto \top]) = ?$$
$$C(\psi[x_1 \mapsto \top, x_2 \mapsto \bot, \ldots, x_n \mapsto \bot]) = ?$$

Here we need another heuristic.

Variable Instantiation Order - Preference Function

First we fix an instantiation order for every single variable by the notion of *preference function*

$$\mathrm{Pref}(x_i) = \begin{cases} \top & \text{if } \mathrm{C}(\psi[x_i \mapsto \top]) \leq \mathrm{C}(\psi[x_i \mapsto \bot]) \\ \bot & \text{otherwise} \end{cases}$$

For a variable x, $\mathrm{Pref}(x)$ returns the Boolean assignment which has the lower cost.

For example,
If $\mathrm{C}(\psi[x_i \mapsto \top]) = 10$ and $\mathrm{C}(\psi[x_i \mapsto \bot]) = 3$,
then $\mathrm{Pref}(x_i) = \bot$.

Variable Instantiation Order - New Algorithm

We use Pref to modify the evaluation function $[\![\cdot]\!]$ as follows.

$$
\begin{aligned}
[\![\forall x\ \varphi]\!] &:= [\![\varphi[x \mapsto \mathrm{Pref}(x)]]\!] \ \wedge\ [\![\varphi[x \mapsto \neg\mathrm{Pref}(x)]]\!] \\
[\![\forall \bar{x}\ \varphi]\!] &:= [\![\forall(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \mathrm{Pref}(x_1)])]\!] \\
&\quad\ \wedge\ [\![\forall(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \neg\mathrm{Pref}(x_1)])]\!] \\
[\![\exists x\ \varphi]\!] &:= [\![\varphi[x \mapsto \mathrm{Pref}(x)]]\!] \ \vee\ [\![\varphi[x \mapsto \neg\mathrm{Pref}(x)]]\!] \\
[\![\exists \bar{x}\ \varphi]\!] &:= [\![\exists(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \mathrm{Pref}(x_1)])]\!] \\
&\quad\ \vee\ [\![\exists(\bar{x} \setminus x_1)(\varphi[x_1 \mapsto \neg\mathrm{Pref}(x_1)])]\!].
\end{aligned}
$$

Variable Instantiation Order

In this algorithm, variable order induces instantiation order.

Example

$$\varphi = \forall x_1, x_2, x_3 \psi$$

where $\mathrm{Pref}(x_1) = \top$, $\mathrm{Pref}(x_2) = \bot$, and $\mathrm{Pref}(x_3) = \bot$.

Instantiation Order		
$\{ [x_1 \mapsto \top]$,	$[x_2 \mapsto \bot]$,	$[x_3 \mapsto \bot] \}$
$\{ [x_1 \mapsto \top]$,	$[x_2 \mapsto \bot]$,	$[x_3 \mapsto \top] \}$
$\{ [x_1 \mapsto \top]$,	$[x_2 \mapsto \top]$,	$[x_3 \mapsto \bot] \}$
	$\ldots$	
$\{ [x_1 \mapsto \bot]$,	$[x_2 \mapsto \top]$,	$[x_3 \mapsto \top] \}$

Variable Instantiation Order

We introduce the significance of variable x_i as

$$\text{Diff}(x_i) = \text{C}(\psi[x_i \mapsto \neg\text{Pref}(x_i)]) - \text{C}(\psi[x_i \mapsto \text{Pref}(x_i)]).$$

For example,
If $\text{C}(\psi[x_i \mapsto \top]) = 10$ and $\text{C}(\psi[x_i \mapsto \bot]) = 3$,
then $\text{Pref}(x_i) = \bot$ and $\text{Diff}(x_i) = 7$.

By Sorting $\bar{x}$ by Diff in descending order, we can have better instantiation order.

Variable Instantiation Order - Example

	x_1	x_2	x_3
$C(\psi[x_i \mapsto \top])$	3	4	5
$C(\psi[x_i \mapsto \bot])$	2	7	3

Variable Instantiation Order - Example

	x_1	x_2	x_3
$C(\psi[x_i \mapsto \top])$	3	$\underline{4}$	5
$C(\psi[x_i \mapsto \bot])$	$\underline{2}$	7	$\underline{3}$
$\mathrm{Diff}(x_i)$	1	3	2

Pref is marked and Diff is calculated.

Variable Instantiation Order - Example

	x_2	x_3	x_1
$C(\psi[x_i \mapsto \top])$	$\underline{4}$	5	3
$C(\psi[x_i \mapsto \bot])$	7	$\underline{3}$	$\underline{2}$
$\mathrm{Diff}(x_i)$	3	2	1

$\bar{x}$ is sorted by Diff.

Instantiation Order			Cost
$\{ [x_2 \mapsto \top],$	$[x_3 \mapsto \bot],$	$[x_1 \mapsto \bot] \}$	3
$\{ [x_2 \mapsto \top],$	$[x_3 \mapsto \bot],$	$[x_1 \mapsto \top] \}$	3.3
$\{ [x_2 \mapsto \top],$	$[x_3 \mapsto \top],$	$[x_1 \mapsto \bot] \}$	3.6
	$\dots$		$\dots$
$\{ [x_2 \mapsto \bot],$	$[x_3 \mapsto \top],$	$[x_1 \mapsto \top] \}$	5

Experiment - Configuration

Almost the same configuration with QBFEVAL'08

- CPU : Intel Pentium D 3.0 GHz
- RAM : 4GB
- OS : Ubuntu/GNU Linux 7.10
- Timeout : 600 sec

Experiment - Test Suite

- QBF 1.0 benchmark from QBFLIB
- 92 instances of non-prenex and non-CNF QBF
- generated from the computation of state space diameters of four models:
  - chain of inverters (Ring, 20 instances)
  - semaphore-based mutual exclusion protocol (Semaphore, 16 instances)
  - chain of inverters (Counter, 45 instances)
  - distributed mutual exclusion protocol (DME, 11 instances)

State-of-the-art Solvers

Three state-of-the-art solvers are chosen:

- QuBE6.0
- sKizzo v0.8.2-beta
- Quantor 3.0 with Picosat-632.

They only accept prenex and CNF instances. We used the converter provided by QBFLIB and did not include conversion time in running time.

Comparison Result

Model	**Quantor**	**sKizzo**	**QuBE**	***CMS***
Ring	10/20	11/20	14/20	20/20
Semaphore	16/16	14/16	13/16	16/16
Counter	28/45	33/45	29/45	27/45
DME	0/11	0/11	6/11	0/11
Total	54/92	58/92	62/92	63/92

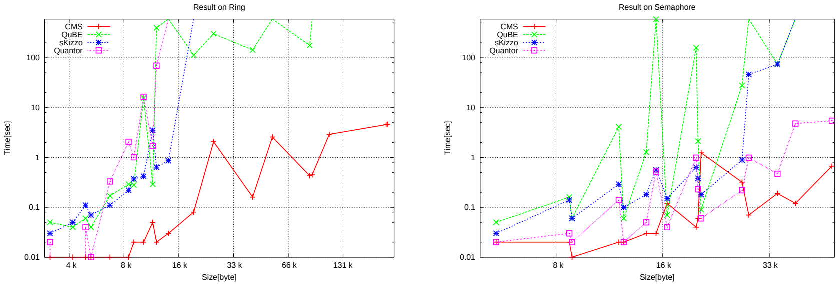
Table: Number of solved instances

Comparison Result

Model	**Quantor**	**sKizzo**	**QuBE**	***CMS***
Ring	77,402	91,780	250,351	973,093
Semaphore	336,596	248,583	219,908	336,596
Counter	341,074	707,679	409,578	308,826
DME	0	0	633,770	0
Total	755,072	1,048,042	1,513,607	1,618,515

Table: Total size of solved instances (in bytes)

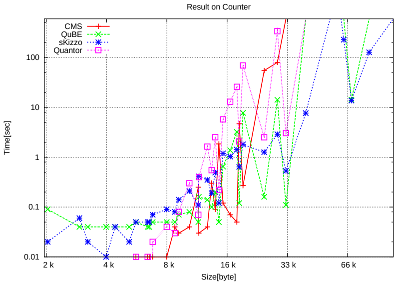
Comparison Result



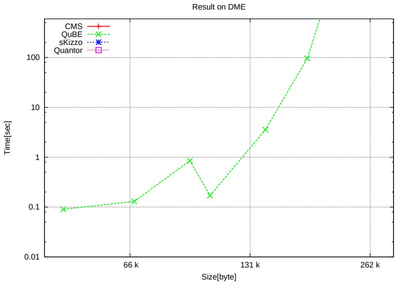
(a) Ring (b) Semaphore

Figure: Performance comparison between *CMS* and three state-of-the-art solvers on QBF 1.0 test suite

Comparison Result



(c) Counter



(d) DME

Figure: Performance comparison between *CMS* and three state-of-the-art solvers on QBF 1.0 test suite

Future Work

- Using better search algorithm
  such as A^* instead of depth-first search.
- Using BDD instead of tree representation.
- Extending to support more logical operator
  such as XOR and EQUIV which frequently occur in
  electronic design.

References

Giunchiglia, E., Narizzano, M., Tacchella, A.:
Backjumping for quantified boolean logic satisfiability.
In: IJCAI. (2001) 275–281

Zhang, L., Malik, S.:
Conflict driven learning in a quantified boolean satisfiability solver.
In: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design (ICCAD'02), New York, NY, USA, ACM (2002) 442–449

Benedetti, M.:
Evaluating QBFs via symbolic Skolemization.
In: 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR04). Number 3452 in LNCS.

Biere, A.:
Resolve and expand.
In: Proceedings of 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Springer-Verlag (2004)

Benedetti, M.:
sKizzo: A QBF decision procedure based on propositional skolemization and symbolic reasoning.
Technical report, ITC-Irst (2004)

Tseytin, G.S.:
On the complexity of derivation in propositional calculus.
In: Studies in Constructive Mathematics and Mathematical Logic II. Volume 8 of Zapiski Nauchnykh Seminarov LOMI. Nauka, Leningrad (1968) 235–259 In Russian.

- Giunchiglia, E., Narizzano, M., Tacchella, A.:
  QuBE: A system for deciding quantified boolean formulas satisfiability.
  In: Proceedings of 1st International Joint Conference on Automated Reasoning (IJCAR'01). (2001) 364–369
- Giunchiglia, E., Narizzano, M., Tacchella, A.:
  Quantified Boolean Formulas Satisfiability Library (QBFLIB) (2001) `www.qbflib.org`.
- Peschiera, C., Pulina, L., Tacchella, A.:
  QBF solver evaluation portal
  `http://www.qbflib.org/qbfeval`.

Thank you

Question?