Quantified Boolean Formula: Evaluation, Certification, and Applications

Jie-Hong Roland Jiang
National Taiwan University

TAROT Summer School
Saint Petersburg, Russia, June 2011
Outline

- Satisfiability (SAT)
  - Conjunctive Normal Form (CNF)
  - SAT solving and Craig interpolation
  - Application
    - Functional dependency

- Quantified Satisfiability (QSAT)
  - Quantified Boolean Formula (QBF)
  - QBF evaluation and certification
  - Application
    - Relation determinization, program synthesis
Satisfiability
Normal Forms

- A **literal** is a variable or its negation
- A **clause (cube)** is a disjunction (conjunction) of literals
- A **conjunctive normal form (CNF)** is a conjunction of clauses; a **disjunctive normal form (DNF)** is a disjunction of cubes

**E.g.,**

- CNF: \((a+b+c)(a+c)(b+d)(\neg a)\)
  - \((\neg a)\) is a unit clause, \(d\) is a pure literal
- DNF: \(a\neg bc + a\neg c + bd + \neg a\)
Satisfiability

- The **satisfiability** (SAT) problem asks whether a given CNF formula can be true under some assignment to the variables.

- In theory, SAT is intractable.
  - The first shown NP-complete problem [Cook, 1971].

- In practice, modern SAT solvers work ‘mysteriously’ well on application CNFs with ~100,000 variables and ~1,000,000 clauses.
  - It enables various applications, and inspires QBF and SMT (Satisfiability Modulo Theories) solver development.
SAT Competition

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

CPU Time (in seconds)

Number of problems solved

http://www.satcompetition.org/PoS11/
SAT Solving

- Ingredients of modern SAT solvers:
  - DPLL-style search
    - [Davis, Putnam, Logemann, Loveland, 1962]
  - Conflict-driven clause learning (CDCL)
    - [Marques-Silva, Sakallah, 1996 (GRASP)]
  - Boolean constraint propagation (BCP) with two-literal watch
    - [Moskewicz, Modigan, Zhao, Zhang, Malik, 2001 (Chaff)]
  - Decision heuristics using variable activity
    - [Moskewicz, Modigan, Zhao, Zhang, Malik, 2001 (Chaff)]
  - Restart
  - Preprocessing
  - Support for incremental solving
    - [Een, Sorensson, 2003 (MiniSat)]
Algorithm $\text{DPLL}(\Phi)$
{
    while there is a unit clause $\{l\}$ in $\Phi$
        $\Phi = \text{BCP}(\Phi, l)$;
    while there is a pure literal $l$ in $\Phi$
        $\Phi = \text{assign}(\Phi, l)$;
    if all clauses of $\Phi$ satisfied  return $\text{true}$;
    if $\Phi$ has a conflicting clause  return $\text{false}$;
    $l := \text{choose_literal}(\Phi)$;
    return $\text{DPLL}(\text{assign}(\Phi, \neg l)) \lor \text{DPLL}(\text{assign}(\Phi, l))$;
}
DPLL Procedure

- Chorological backtrack

- E.g.

```
~a ~b b ~c c d
{~a,e} {a,b,~c} {c,~d} {a,b,d} {d,e} {c,d,~e}
```

![Diagram](image.png)
Modern SAT Procedure

Algorithm $\text{CDCL}(\Phi)$
{
    while (1)
        while there is a unit clause \{l\} in $\Phi$
            $\Phi = \text{BCP}(\Phi, l)$;
        while there is a pure literal l in $\Phi$
            $\Phi = \text{assign}(\Phi, l)$;
        if $\Phi$ contains no conflicting clause
            if all clauses of $\Phi$ are satisfied $\text{return true;}$
                l := \text{choose_literal}(\Phi);
                assign(\Phi, l);
            else
                if conflict at top decision level $\text{return false;}$
                    analyze_conflict();
                    undo assignments;
                    $\Phi := \text{add_conflict_clause}(\Phi)$;
    }
}
Conflict Analysis & Clause Learning

- There can be many learnt clauses from a conflict
- Clause learning admits non-chronological backtrack

E.g.,
\{-\neg x_{10587}, \neg x_{10588}, \neg x_{10592}\}
...
\{-\neg x_{10374}, \neg x_{10582}, \neg x_{10578}, \neg x_{10373}, \neg x_{10629}\}
...
\{x_{10646}, x_{9444}, \neg x_{10373}, \neg x_{10635}, \neg x_{10637}\}
Clause Learning as Resolution

- **Resolution** of two clauses $C_1 \lor x$ and $C_2 \lor \neg x$:

\[
\begin{array}{c}
C_1 \lor x & C_2 \lor \neg x \\
\hline
C_1 \lor C_2
\end{array}
\]

where $x$ is the **pivot variable** and $C_1 \lor C_2$ is the **resolvant**, i.e., $C_1 \lor C_2 = \exists x.(C_1 \lor x)(C_2 \lor \neg x)$

- A learnt clause can be obtained from a sequence of resolution steps
  - Exercise:
    - Find a resolution sequence leading to the learnt clause
      $\{ \neg x10374, \neg x10582, \neg x10578, \neg x10373, \neg x10629 \}$ in the previous slides
Resolution

Resolution is complete for SAT solving
- A CNF formula is unsatisfiable if and only if there exists a resolution sequence leading to the empty clause

Example

\[(a \lor b \lor c)(\neg a \lor c)(\neg b \lor \neg d)(\neg c)(c \lor d)\]

\[(b \lor c)\]

\[(d)\]

\[(c \lor \neg d)\]

\[\neg d\]

\[()\]
SAT Certification

- True CNF
  - Satisfying assignment (model)
    - Verifiable in linear time

- False CNF
  - Resolution refutation
    - Potentially of exponential size
Craig Interpolation

[Craig Interpolation Thm, 1957]
If $A \land B$ is UNSAT for formulae $A$ and $B$, there exists an interpolant $I$ of $A$ such that

1. $A \Rightarrow I$
2. $I \land B$ is UNSAT
3. $I$ refers only to the common variables of $A$ and $B$

$I$ is an abstraction of $A$
Interpolant and Resolution Proof

- SAT solver may produce the resolution proof of an UNSAT CNF $\varphi$
- For $\varphi = \varphi_A \land \varphi_B$ specified, the corresponding interpolant can be obtained in time linear in the resolution proof

\[
\varphi_A = (a \lor b \lor c)(\neg a \lor c)(\neg b \lor \neg d)(\neg c)(c \lor d)
\]
\[
\varphi_B = (b \lor c)(c)(1)(1)(1)
\]

\[
\neg d
\]

\[
\frac{2011/6/29}{TAROT 2011}
\]

[McMillan, 2003]
Circuit to CNF Conversion

- Circuit to CNF conversion can be done in time linear w.r.t. circuit size [Tseitin, 1968]
  - Trick: introduce intermediate variables
    - The resultant formula can blow up if no intermediate variables are allowed to exist
Circuit to CNF Conversion

Example

- Single gate:

\[ (\neg a + \neg b + c)(a + \neg c)(b + \neg c) \]

- Circuit of connected gates:

\[ (\neg 1 + 2 + 4)(1 + \neg 4)(\neg 2 + \neg 4) \]
\[ (\neg 2 + \neg 3 + 5)(2 + \neg 5)(3 + \neg 5) \]
\[ (2 + \neg 3 + 6)(\neg 2 + \neg 6)(3 + \neg 6) \]
\[ (\neg 4 + \neg 5 + 7)(4 + \neg 7)(5 + \neg 7) \]
\[ (5 + 6 + 8)(\neg 5 + \neg 8)(\neg 6 + \neg 8) \]
\[ (7 + 8 + 9)(\neg 7 + \neg 9)(\neg 8 + \neg 9) \]
\[ (9) \]

Is output always 0?
Justify to "1"
 SAT Application
 Functional Dependency

- $f(x)$ **functionally depends** on $g_1(x), g_2(x), ..., g_m(x)$ if $f(x) = h(g_1(x), g_2(x), ..., g_m(x))$, denoted $h(G(x))$

  - Under what condition can function $f$ be expressed as some function $h$ over a set of given functions $G={g_1,...,g_m}$?
  - $h$ exists $\iff \nexists a,b$ such that $f(a) \neq f(b)$ and $G(a)=G(b)$

  i.e., $G$ is more distinguishing than $f$
Applications of functional dependency
- Resynthesis/rewiring
- Redundant register removal
- BDD minimization
- Verification reduction
- ...

![Boolean Network Diagram]

- target function
- base functions
SAT Application
Functional Dependency

- Computing \( h \)
  
  \[
  h^{\text{on}} = \{y \in B^m : y = G(x) \text{ and } f(x) = 1, \ x \in B^n\}
  \]
  
  \[
  h^{\text{off}} = \{y \in B^m : y = G(x) \text{ and } f(x) = 0, \ x \in B^n\}
  \]
SAT Application
Functional Dependency

- $h$ exists $\iff$ 
  $\not\exists a, b$ such that $f(a) \neq f(b)$ and $G(a) = G(b)$, 
  i.e., $(f(x) \neq f(x^*)) \land (G(x) \equiv G(x^*))$ is UNSAT

- How to derive $h$? How to select $G$?
(f(x) \neq f(x^*)) \land (G(x) \equiv G(x^*)) \text{ is UNSAT}
SAT Application
Functional Dependency

- Clause set $A$: $C_{DFN_{on}}$, $y_0$
- Clause set $B$: $C_{DFN_{off}}$, $\neg y_0^*$, ($y_i = y_i^*$) for $i = 1, \ldots, m$
- $I$ is an overapproximation of $Img(f_{on})$ and is disjoint from $Img(f_{off})$
- $I$ only refers to $y_1, \ldots, y_m$
- Therefore, $I$ corresponds to a feasible implementation of $h$

[Lee, J, Huang, Mishchenko, 2007]
Quantified Satisfiability
Quantified Boolean Formula

A quantified Boolean formula (QBF) is often written in **prenex form** (with quantifiers placed on the left) as

\[ Q_1 x_1, \ldots, Q_n x_n. \varphi \]

for \( Q_i \in \{ \forall, \exists \} \) and \( \varphi \) a quantifier-free formula

- If \( \varphi \) is further in CNF, the corresponding QBF is in the so-called **prenex CNF** (PCNF), the most popular QBF representation
- Any QBF can be converted to PCNF
Quantified Boolean Formula

- Quantification order matters in a QBF
- A variable $x_i$ in $(Q_1 x_1,\ldots, Q_i x_i,\ldots, Q_n x_n. \varphi)$ is of level $k$ if there are $k$ quantifier alternations (i.e., changing from $\forall$ to $\exists$ or from $\exists$ to $\forall$) from $Q_1$ to $Q_i$.

Example

$$\forall a \ \exists b \ \forall c \ \forall d \ \exists e. \varphi$$

$\text{level}(a)=0$, $\text{level}(b)=1$, $\text{level}(c)=2$, $\text{level}(d)=2$, $\text{level}(e)=3$
Quantified Boolean Formula

- Many decision problems can be compactly encoded in QBFs

- In theory, QBF solving (QSAT) is PSPACE complete
  - The more the quantifier alternations, the higher the complexity in the Polynomial Hierarchy

- In practice, solvable QBFs are typically of size ~1,000 variables
QBF Solver

- QBF solver choices
  - Data structures for formula representation
    - Prenex vs. non-prenex
    - Normal form vs. non-normal form
      - CNF, NNF, BDD, AIG, etc.
  - Solving mechanisms
    - Search, Q-resolution, Skolemization, quantifier elimination, etc.
  - Preprocessing techniques

- Standard approach
  - Search-based PCNF formula solving (similar to SAT)
    - Both clause learning (from a conflicting assignment) and cube learning (from a satisfying assignment) are performed
QBF Solving

Example

\[ \exists a \forall x \exists b \forall y \exists c \quad (a + b + y + c)(a + x + b + y + c)(x + b)(y + c)(c + a + x + b)(x + b)(a + b + y) \]

\[ \begin{align*}
&\ll a, L \gg \\
&= (b + y + c)(x + b + y + c)(x + b)(y + c)(x + b)(b + y) \\
&\ll x, L \gg \\
&= (b + y + c)(b + y + c)(y + c)(b + y) \\
&\ll b, U \gg \\
&= (y + c)(y + c)(y + c) \\
&\ll y, L \gg \\
&= (c)(c) \\
&\{false\} \\
&\ll c, R \gg \\
&= (c) \\
&\{true\} \\
&\ll x, R \gg \\
&= (x + b)(y + c)(c + x + b)(x + b) \\
&\ll y, P \gg \\
&= (x + b)(c + x + b)(x + b) \\
&\ll c, U \gg \\
&= (x + b)(x + b)(x + b) \\
&\ll x, L \gg \\
&= (b) \\
&\{true\} \\
&\ll x, R \gg \\
&= (b)(b) \\
&\{false\} \\
\end{align*} \]
Q-Resolution

- **Q-resolution** on PCNF is similar to resolution on CNF, except that the pivots are restricted to existentially quantified variables and the additional rule of \( \forall\text{-reduction} 
\)

\[
\begin{array}{c}
C_1 \lor x \\
\hline
C_2 \lor \neg x
\end{array}
\]

\( \forall\text{-RED}(C_1 \lor C_2) \)

where operator \( \forall\text{-RED} \) removes from \( C_1 \lor C_2 \) the universally (\( \forall \)) quantified variables whose quantification levels are greater than any of the existentially (\( \exists \)) quantified variables in \( C_1 \lor C_2 \)

- E.g.,
  - prefix: \( \forall a \exists b \forall c \forall d \exists e \)
  - \( \forall\text{-RED}(a+b+c+d) = (a+b) \)

- Q-resolution is complete for QBF solving
  - A PCNF formula is unsatisfiable if and only if there exists a Q-resolution sequence leading to the empty clause
Q-Resolution

Example (cont’d)

\[ \exists a \forall x \exists b \forall y \exists c \quad (a + b + y + c)(a + x + b + y + c)(x + b)(y + c)(c + a + x + b)(x + b)(a + b + y) \]

![Diagram of Q-Resolution process]
Skolemization

- Skolemization and Skolem normal form
  - Existentially quantified variables are replaced with function symbols
  - QBF prefix contains only two quantification levels
    - $\exists$ function symbols, $\forall$ variables

- Example

$$\forall a \exists b \forall c \exists d. \ (\neg a + \neg b)(\neg b + \neg c + \neg d)(\neg b + c + d)(a + b + c)$$

Skolem functions

$$\exists F_b(a) \ \exists F_d(a, c) \ \forall a \ \forall c. \ (\neg a + \neg F_b)(\neg F_b + \neg c + \neg F_d)(\neg F_b + c + F_d)(a + F_b + c)$$
QBF Certification

- QBF certification
  - Ensure correctness and, more importantly, provide useful information
  - Certificates
    - True QBF: term-resolution proof / Skolem-function (SF) model
      - SF model is more useful in practical applications
    - False QBF: clause-resolution proof / Herbrand-function (HF) countermodel
      - HF countermodel is more useful in practical applications

- Solvers and certificates
  - To date, only Skolemization-based solvers (e.g., sKizzo, squolem, Ebddres) can provide SFs
  - Search-based solvers (e.g., QuBE) are the most popular and can be instrumented to provide resolution proofs
QBF Certification

- Solvers and certificates

<table>
<thead>
<tr>
<th>Solver</th>
<th>Algorithm</th>
<th>Certificate</th>
<th>True QBF</th>
<th>False QBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuBE-cert</td>
<td>search</td>
<td>Cube resolution</td>
<td>Clause resolution</td>
<td></td>
</tr>
<tr>
<td>yQuaffle</td>
<td>search</td>
<td>Cube resolution</td>
<td>Clause resolution</td>
<td></td>
</tr>
<tr>
<td>Ebddres</td>
<td>Skolemization</td>
<td>Skolem function</td>
<td>Clause resolution</td>
<td></td>
</tr>
<tr>
<td>sKizzo</td>
<td>Skolemization</td>
<td>Skolem function</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>squolem</td>
<td>Skolemization</td>
<td>Skolem function</td>
<td>Clause resolution</td>
<td></td>
</tr>
</tbody>
</table>
QBF Certification

- Incomplete picture of QBF certification

<table>
<thead>
<tr>
<th></th>
<th>Syntactic Certificate</th>
<th>Semantic Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>True QBF</td>
<td>Cube-resolution proof</td>
<td>Skolem-function model</td>
</tr>
<tr>
<td>False QBF</td>
<td>Clause-resolution proof</td>
<td>?</td>
</tr>
</tbody>
</table>

- Recent progress
  - Herbrand-function countermodel
    - [Balabanov, J, 2011 (ResQu)]
  - Syntactic to semantic certificate conversion
    - Linear time [Balabanov, J, 2011 (ResQu)]
QBF Certification

- Unified QBF certification

- True QBF
  - Cube resolution proof
  - Skolem function (model)
  - ResQu

- False QBF
  - Clause resolution proof
  - Herbrand function (countermodel)
  - ResQu

formula negation
A Skolem-function model (Herbrand-function countermodel) for a true (false) QBF can be derived from its cube (clause) resolution proof.

A **Right-First-And-Or (RFAO) formula** is recursively defined as follows.

\[ \varphi ::= \text{clause} \mid \text{cube} \mid \text{clause} \land \varphi \mid \text{cube} \lor \varphi \]

E.g.,

\[ (a'+b) \land ac \lor (b'+c') \land bc \]

\[ = ((a'+b) \land (ac \lor ((b'+c') \land bc))) \]

**ResQu**
ResQu

countermodel_construct
input: a false QBF $\Phi$ and its clause-resolution DAG $G_H(V_H, E_H)$
output: a countermodel in RFAO formulas
begin
01 foreach universal variable $x$ of $\Phi$
02 RFAO_node_array[$x$] := $\emptyset$;
03 foreach vertex $v$ of $G_H$ in topological order
04 if $v$.clause resulted from $\forall$-reduction on $u$.clause, i.e., $(u, v) \in E_H$
05 $v$.cube := $\neg(v$.clause$)$;
06 foreach universal variable $x$ reduced from $u$.clause to get $v$.clause
07 if $x$ appears as positive literal in $u$.clause
08 push $v$.clause to RFAO_node_array[$x$];
09 else if $x$ appears as negative literal in $u$.clause
10 push $v$.cube to RFAO_node_array[$x$];
11 if $v$.clause is the empty clause
12 foreach universal variable $x$ of $\Phi$
13 simplify RFAO_node_array[$x$];
14 return RFAO_node_array's;
end
ResQu

Example

\[ \exists a \forall x \exists b \forall y \exists c \]

\[(a + b + y + c)_1 (a + x + b + y + \bar{c})_2 (x + \bar{b})_3 (y + c)_4 (\bar{a} + \bar{x} + b + \bar{c})_5 (x + \bar{b})_6 (a + \bar{b} + \bar{y})_7 \]

\[ (a + x + b + y)_8 \]

(2) \[ (a + x + b + y)_{8+} \]

(3) \[ (a + x)_{9} \]

(4) \[ (\bar{a} + x + b + y)_{10} \]

(5) \[ (\bar{a} + x + b)_{10+} \]

(6) \[ (a + x)_{11} \]

(1) \[ (a + \bar{b})_{7+} \]

\[ (empty) \]

0. \[ x: [] \quad y: [] \]

1. \[ x: [] \quad y: [\text{cube}(\bar{a}b)] \]

2. \[ x: [] \quad y: [\text{cube}(\bar{a}b), \text{clause}(a + x + b)] \]

3. \[ x: [\text{clause}(a)] \quad y: [\text{cube}(\bar{a}b), \text{clause}(a + x + b)] \]

4. \[ x: [\text{clause}(a)] \quad y: [\text{cube}(\bar{a}b), \text{clause}(a + x + b), \text{cube}(a\bar{x}b)] \]

5. \[ x: [\text{clause}(a), \text{cube}(a)] \quad y: [\text{cube}(\bar{a}b), \text{clause}(a + x + b), \text{cube}(a\bar{x}b)] \]
QBF Certification

- Applications of Skolem/Herbrand functions
  - Program synthesis
  - Winning strategy synthesis in two player games
  - Plan derivation in AI
  - Logic synthesis
  - ...

2011/6/29
QBF Application
Relation Determinization

- **Relation** $R(X, Y)$
  - Allow one-to-many mappings
  - Can describe non-deterministic behavior
  - More generic than functions

- **Function** $F(X)$
  - Disallow one-to-many mappings
  - Can only describe deterministic behavior
  - A special case of relation

<table>
<thead>
<tr>
<th>$x_1x_2$</th>
<th>$y_1y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1x_2$</th>
<th>$y_1y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

$f_1 = x_1x_2$
$f_2 = \neg x_1 \neg x_2$
QBF Application
Relation Determinization

- **Total relation**
  - Every input element is mapped to at least one output element

- **Partial relation**
  - Some input element is not mapped to any output element

\[ x_1 x_2 \]
\[ \begin{array}{c}
  00 \\
  01 \\
  10 \\
  11 \\
\end{array} \rightarrow \begin{array}{c}
  0 \\
  1 \\
\end{array} \]

\[ x_1 x_2 \]
\[ \begin{array}{c}
  00 \\
  01 \\
  10 \\
  11 \\
\end{array} \rightarrow \begin{array}{c}
  0 \\
  1 \\
\end{array} \]
A partial relation can be **totalized**

Assume that the input element not mapped to any output element is a don’t care

\[ T(X, y) = R(X, y) \lor \forall y. \neg R(X, y) \]
Applications of Boolean relation

- In high-level design, Boolean relations can be used to describe (nondeterministic) specifications.
- In gate-level design, Boolean relations can be used to characterize the flexibility of sub-circuits.

- Boolean relations are more powerful than traditional don’t-care representations.
QBF Application
Relation Determinization

- Relation determinization
  - For hardware implementation of a system, we need functions rather than relations
    - Hardware systems are intrinsically deterministic
      - One input stimulus results in one output response
  - To simplify implementation, we can explore the flexibilities described by a relation for optimization
QBF Application
Relation Determinization

Example

\[ f_1 = x_1 x_2 \]
\[ f_2 = \neg x_1 \neg x_2 \]
QBF Application
Relation Determinization

Given a *nondeterministic* Boolean relation $R(X, Y)$, how to determinize and extract functions from it?

Solve QBF

$$\forall x_1, \ldots, \forall x_m, \exists y_1, \ldots, \exists y_n. R(x_1, \ldots, x_m, y_1, \ldots, y_n)$$

The Skolem functions of variables $y_1, \ldots, y_n$ correspond to the output-functions we want.
Program synthesis by sketching

[Solar-Lezama et al., 2006]

Example

Spec:
```c
int foo (int x){
    return x+x;
}
```

Sketch:
```c
int bar (int x) implements foo{
    return x << ??;
}
```

Result:
```c
int bar (int x) implements foo{
    return x << 1;
}
```
QBF Application
Program Synthesis

Sketch synthesis can be solved by searching for control values satisfying

\[ \exists c \forall x. \ Spec(x) = Sk(x,c) \]

We are interested to derive the Skolem function (in this case, constant) of c
Conclusions

- Modern SAT/QSAT solvers are powerful tools for solving large-scale synthesis, verification, and other computer science problems.

- Certificates of SAT/QSAT solving may be utilized to extract essential information for applications in synthesis and verification.

- Understanding how solvers work helps practitioners formulate and solve real-world problems.
Suggested Further Exploration

- SMT solvers and their applications in program analysis and verification
Contributors

- Valeriy Balabanov, NTU
- Wei-Lun Hung, NTU
- Chih-Chun Lee, NTU
- Hsuan-Po Lin, NTU
- Alan Mishchenko, UC Berkeley
Thank You!

Questions?