What is the State-of-the-Art in DQBF solving?

Gergely Kovásznai

IoT Research Institute,
Eszterhazy Karoly University of Applied Sciences,
Eger, Hungary
kovasznai.gergely@iot.uni-eger.hu

MACS 2016
May 21, 2016
Eger, Hungary
What is this all about?

Formal (a priori) verification of systems.

- Hardware = logical circuits

- Software

What is the State-of-the-Art in DQBF solving?
Let’s start with SAT solving!

Boolean formulas, SAT:

\[(e \lor \overline{u}) \land (\overline{e} \lor u)\]

SAT technology - Applied in our life [le Berre et al., 2011]:
- Intel Core i7 CPUs designed with the help of SAT solvers
- Windows device drivers verified on SAT bases (SMT solver, Z3)

Millions of variables and clauses.

Regular SAT-related competitions:
- SAT Competition 2016, SAT Race 2015
- SMT Competition 2016
- Hardware Model Checking Competition 2015
Let’s start with SAT solving!

Boolean formulas, SAT:

\[(e \lor \overline{u}) \land (\overline{e} \lor u)\]

SAT technology - Applied in our life [le Berre et al., 2011]:

- Intel Core i7 CPUs designed with the help of SAT solvers
- Windows device drivers verified on SAT bases (SMT solver, Z3)

Millions of variables and clauses.

Regular SAT-related competitions:

- SAT Competition 2016, SAT Race 2015
- SMT Competition 2016
- Hardware Model Checking Competition 2015

Gergely Kovásznai
What is the State-of-the-Art in DQBF solving?
Let’s start with SAT solving!

Boolean formulas, SAT:

\[(e \lor \overline{u}) \land (\overline{e} \lor u)\]

SAT technology - Applied in our life [le Berre et al., 2011]:
- Intel Core i7 CPUs designed with the help of SAT solvers
- Windows device drivers verified on SAT bases (SMT solver, Z3)

Millions of variables and clauses.

Regular SAT-related competitions:
- SAT Competition 2016, SAT Race 2015
- SMT Competition 2016
- Hardware Model Checking Competition 2015
What is the State-of-the-Art in DQBF solving?
**QBF – Generalizing SAT**

Boolean formulas, SAT:

\[(e \lor \overline{u}) \land (\overline{e} \lor u)\]

Quantified Boolean Formulas:

\[\exists e \forall u . (e \lor \overline{u}) \land (\overline{e} \lor u)\]

Applications: model checking, formal verification or synthesis, etc.

QBF is not yet widely applied in industry, but is evolving rapidly.

- Several solvers such as Quantor, DepQBF, RAReQS, etc.
- Preprocessors such as Bloqer, QxBF, Qprocessor
- Proof extractors and validators such as QBFcert

Ten thousands of variables and hundred thousands of clauses.

QBF competitions: QBF-EVAL 2016, QBF Gallery 2014
QBF – Generalizing SAT

Boolean formulas, SAT:

\[(e \lor \overline{u}) \land (\overline{e} \lor u)\]

Quantified Boolean Formulas:

\[\exists e \forall u . (e \lor \overline{u}) \land (\overline{e} \lor u)\]

Applications: model checking, formal verification or synthesis, etc.

QBF is not yet widely applied in industry, but is evolving rapidly.

- Several solvers such as Quantor, DepQBF, RAReQS, etc.
- Preprocessors such as Bloqer, QxBF, Qprocessor
- Proof extractors and validators such as QBFcert

Ten thousands of variables and hundred thousands of clauses.

QBF competitions: QBFEVAL 2016, QBF Gallery 2014
Boolean formulas, SAT:

\((e \lor \overline{u}) \land (\overline{e} \lor u)\)

Quantified Boolean Formulas:

\(\exists e \forall u . (e \lor \overline{u}) \land (\overline{e} \lor u)\)

Applications: model checking, formal verification or synthesis, etc.

QBF is not yet widely applied in industry, but is evolving rapidly.
- Several solvers such as Quantor, DepQBF, RAReQS, etc.
- Preprocessors such as Bloqer, QxBF, Qprocessor
- Proof extractors and validators such as QBFcert

Ten thousands of variables and hundred thousands of clauses.

QBF competitions: QBFEVAL 2016, QBF Gallery 2014
Henkin quantifiers are used to express the “independence” of variables from each other.

In QBF: An existential variable depends on all the universal variables to the left in the quantifier prefix.

\[
\forall u_1, u_2 \exists e \forall u_3 \exists f . \ (u_2 \lor u_3 \lor e) \land (u_1 \lor u_2 \lor \overline{e} \lor f)
\]

In DQBF: Variable dependencies are explicitly given.

\[
\forall u_1, u_2, u_3 \exists e(u_1, u_3), f(u_2, u_3) . \ (u_2 \lor u_3 \lor e) \land (u_1 \lor u_2 \lor \overline{e} \lor f)
\]

Fundamental application: partial/imperfect information games.

Higher complexity:
- SAT – NP-complete
- QBF – \text{PSPACE}\text{-complete}
- DQBF – \text{NExpTime}\text{-complete}
Henkin quantifiers are used to express the “independence” of variables from each other.

In QBF: An existential variable depends on all the universal variables to the left in the quantifier prefix.

\[
\forall u_1, u_2 \exists e \forall u_3 \exists f \cdot (u_2 \lor \overline{u}_3 \lor e) \land (u_1 \lor \overline{u}_2 \lor \overline{e} \lor f)
\]

In DQBF: Variable dependencies are explicitly given.

\[
\forall u_1, u_2, u_3 \exists e(u_1, u_3), f(u_2, u_3) \cdot (u_2 \lor \overline{u}_3 \lor e) \land (u_1 \lor \overline{u}_2 \lor \overline{e} \lor f)
\]

Fundamental application: partial/imperfect information games.

Higher complexity:
- SAT – NP-complete
- QBF – PSPACE-complete
- DQBF – NEXPTime-complete
Henkin quantifiers are used to express the “independence” of variables from each other.

In QBF: An existential variable depends on all the universal variables to the left in the quantifier prefix.

\[
\forall u_1, u_2 \ \exists e \ \forall u_3 \ \exists f . \ (u_2 \lor \overline{u}_3 \lor e) \land (u_1 \lor \overline{u}_2 \lor \overline{e} \lor f)
\]

In DQBF: Variable dependencies are explicitly given.

\[
\forall u_1, u_2, u_3 \ \exists e(u_1, u_3), f(u_2, u_3) . \ (u_2 \lor \overline{u}_3 \lor e) \land (u_1 \lor \overline{u}_2 \lor \overline{e} \lor f)
\]

Fundamental application: partial/imperfect information games.

Higher complexity:
- SAT – NP-complete
- QBF – PSPACE-complete
- DQBF – NEXPTIME-complete
1st solving approach – DQDPLL

[Fröhlich, Kovásznai, Biere, 2012]

Adaptation of QDPLL from QBF to DQBF: e.g., unit propagation, clause learning, universal reduction, watched literals, etc.

Implemented, but slow. Why?
1st “killer” application

[Gitina, Reimer, Sauer et al., 2013]
“Killer” app: partial equivalence checking (PEC) of circuits

Black boxes = Existential DQBF vars
Inputs = Universal vars

The goal: To **synthesize** Skolem-functions for the existential vars.
Unrolling the DQBF to QBF. Given a bound $k \geq 1$,

- Use $k$ copies of all variables and the matrix
- Solve the QBF

$$
\forall u_1^1, \ldots, u_m^k \exists e_1^1, \ldots, e_n^k .
$$

$$
consistent(e_1^1, k) \land \cdots \land consistent(e_n^k, k) \Rightarrow \bigvee_{1 \leq i \leq k} \neg \phi_i^k
$$

- Ackermann constraints as a guard:

$$
consistent(e, k) := \bigwedge_{1 \leq i, j \leq k} \left( \bigwedge_{u \in depse} u^i = u^j \Rightarrow e_i = e_j \right)
$$

In practice, it can solve only $UNSAT$ problems.
Adapts and extends the *Inst-Gen* approach to DQBF.

**Inst-Gen:**
- The solving approach for EPR logic
  - The $\exists^* \forall^*.\phi$ fragment of 1st-order logic
  - Has the same complexity as DQBF
- The core of iProver, the most successful EPR-solver
  - Regular theorem proving competition – CASC 2015

*iDQ* solves SAT and UNSAT instances, quite quickly.
Adapts and extends the Inst-Gen approach to DQBF.

Inst-Gen:
- The solving approach for EPR logic
  - The $\exists^*\forall^*.\phi$ fragment of 1st-order logic
  - Has the same complexity as DQBF
- The core of iProver, the most successful EPR-solver
  - Regular theorem proving competition – CASC 2015

iDQ solves SAT and UNSAT instances, quite quickly.
[Gitina, Wimmer, Reimer et al., 2015]
An improved expansion-based solver:

- Expands DQBF to QBF
  - Eliminates (universal and existential) variables

\[
\forall u_1, u_2 \exists e(u_1) . \phi \rightarrow \forall u_2 \exists e, e' . \phi[0/u_1] \land \phi[1/u_1][e'/e]
\]

- Eliminates the **minimum set** of variables that cause non-linear dependencies
  - Expressed as a partial MaxSAT problem

Publicly not available.
# A new solver – HQS

## DQBF PEC benchmarks

<table>
<thead>
<tr>
<th></th>
<th># (sat/uns)</th>
<th>TO/MO</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>adder</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>300 (42/258)</td>
<td>0/0</td>
<td>9.7</td>
</tr>
<tr>
<td>iDQ</td>
<td>216 (3/213)</td>
<td>84/0</td>
<td>89828</td>
</tr>
<tr>
<td><strong>lookahead</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>300 (10/290)</td>
<td>0/0</td>
<td>23.2</td>
</tr>
<tr>
<td>iDQ</td>
<td>273 (4/269)</td>
<td>27/0</td>
<td>39540</td>
</tr>
<tr>
<td><strong>z4</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>240 (72/168)</td>
<td>0/0</td>
<td>4.9</td>
</tr>
<tr>
<td>iDQ</td>
<td>111 (8/103)</td>
<td>129/0</td>
<td>41626</td>
</tr>
<tr>
<td><strong>C432</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>60 (19/41)</td>
<td>0/180</td>
<td>1333</td>
</tr>
<tr>
<td>iDQ</td>
<td>20 (0/20)</td>
<td>85/135</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th># (sat/uns)</th>
<th>TO/MO</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bitcell</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>300 (7/293)</td>
<td>0/0</td>
<td>11.3</td>
</tr>
<tr>
<td>iDQ</td>
<td>190 (2/188)</td>
<td>110/0</td>
<td>78107</td>
</tr>
<tr>
<td><strong>pec_xor</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>200 (24/176)</td>
<td>0/0</td>
<td>33.6</td>
</tr>
<tr>
<td>iDQ</td>
<td>200 (24/176)</td>
<td>0/0</td>
<td>181.6</td>
</tr>
<tr>
<td><strong>comp</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HQS</td>
<td>155 (39/116)</td>
<td>9/76</td>
<td>17.8</td>
</tr>
<tr>
<td>iDQ</td>
<td>25 (0/25)</td>
<td>180/35</td>
<td>11.6</td>
</tr>
</tbody>
</table>

**TO** = timeout  
**MO** = memory out
Advances of DQBF

- Preprocessing [Wimmer, Gitina, Hist et al., 2015]
- Certification [Balabanov, Chiang, Jiang, 2014]
- Resolution calculi [Beyersdorff, Chew, Schmidt, Suda, 2016]
- etc.
Future work

To find new applications for DQBF. Some candidates:

- Synthesis of protocols from existing ones by enforcing some given “template”
- Solving bit-vector formulas (as part of SMT)
- Inference with partial information of network topology
- etc.