A Survey on DQBF: Formulas, Applications, Solving Approaches

Gergely Kovásznai

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

QUANTIFY 2015
August 3, 2015
Berlin, Germany

EGER 1774
ESZTERHAZY KAROLY UNIVERSITY
OF APPLIED SCIENCES

The IF logic

1996: Jaakko Hintikka – Independence Friendly (IF) Logic

- in his book [Jaakko Hintikka. *The Principles of Mathematics Revisited*. 1996.]

Logicians were questioning if IF logic was a logic at all.

- [Janssen. *Independent Choices and the Interpretation of IF Logic*. JLLI, 2002.]
  Strange properties of the IF logic:
  - ϕ, $\phi \wedge \phi$, and $\phi \vee \phi$ are *not* equivalent
  - Bound variables cannot be renamed
- [Feferman. *What Kind of Logic is "Independence Friendly" Logic?*. Library of Living Philosophers, 2006.]
  - Is IF logic a logic at all?

The IF logic

1996: Jaakko Hintikka – Independence Friendly (IF) Logic

- in his book [Jaakko Hintikka. *The Principles of Mathematics Revisited*. 1996.]

Logicians were questioning if IF logic was a logic at all.

- [Janssen. *Independent Choices and the Interpretation of IF Logic*. JLLI, 2002.]

  Strange properties of the IF logic:
  - ϕ, $\phi \wedge \phi$, and $\phi \vee \phi$ are *not* equivalent
  - Bound variables cannot be renamed
- [Feferman. *What Kind of Logic is "Independence Friendly" Logic?*. Library of Living Philosophers, 2006.]
  - Is IF logic a logic at all?

Henkin quantifiers

In the IF logic and in DQBF *Henkin (or branching) quantifiers* are used to express the "independence" of variables from each other.

$$\begin{matrix} \forall x \exists e \\ \forall y \exists f \end{matrix} \left\{ \phi(x, e, y, f) \right.$$

In terms of Skolem functions:

$$\phi\big(x, e(x), y, f(y)\big)$$

In IF logic: ϕ is a 1st-order formula

In DQBF: ϕ is a Boolean formula

Fundamental application:
partial-information (or imperfect-information) games

Henkin quantifiers

In the IF logic and in DQBF *Henkin (or branching) quantifiers* are used to express the "independence" of variables from each other.

$$\begin{matrix} \forall x \exists e \\ \forall y \exists f \end{matrix} \left\{ \phi(x, e, y, f) \right.$$

In terms of Skolem functions:

$$\phi\big(x, e(x), y, f(y)\big)$$

In IF logic: ϕ is a 1st-order formula

In DQBF: ϕ is a Boolean formula

Fundamental application:
partial-information (or imperfect-information) games

Henkin quantifiers

In the IF logic and in DQBF *Henkin (or branching) quantifiers* are used to express the "independence" of variables from each other.

$$\begin{matrix} \forall x \exists e \\ \forall y \exists f \end{matrix} \left\{ \phi(x, e, y, f) \right.$$

In terms of Skolem functions:

$$\phi\big(x, e(x), y, f(y)\big)$$

In IF logic: ϕ is a 1st-order formula

In DQBF: ϕ is a Boolean formula

Fundamental application:
partial-information (or imperfect-information) games

What is DQBF?

[Peterson, Reif. *Multiple-person alternation*. Foundations of Computer Science, 1979.]

- DQBF = *Dependency* Quantified Boolean Formulas

$$\forall u_1, u_2, u_3 \; \exists e(\mathbf{u_1}, \mathbf{u_3}), f(\mathbf{u_2}) \; . \; (u_2 \vee \overline{u}_3 \vee e) \wedge (u_1 \vee \overline{u}_2 \vee \overline{e} \vee f)$$

- Generalization of QBF
- Variable dependencies can be explicitly given
- Higher complexity:
  - QBF – PSpace-complete
  - DQBF – NExpTime-complete

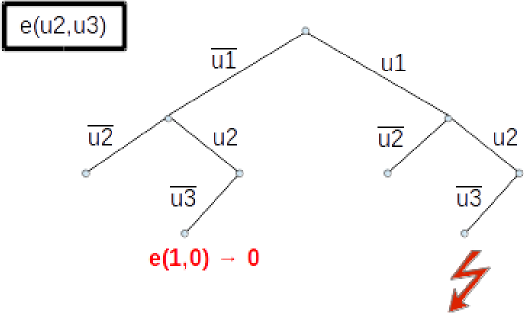
1st solving approach – DQDPLL

[Fröhlich, Kovásznai, Biere. *A DPLL Algorithm for Solving DQBF*. POS, 2012.]

Main motivation: quantifier-free bit-vector formulas $(\mathrm{QF\_BV})$ has the same complexity as DQBF.

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

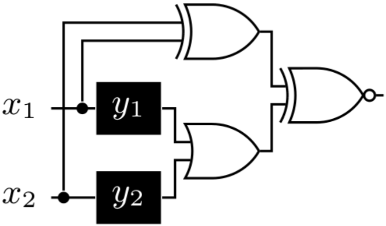
Implemented, but slow. Why?



1st "killer" application

[Gitina, Reimer, Sauer, Wimmer, Scholl, Becker. *Equivalence checking of partial designs using dependency quantified Boolean formulae*. ICCD, 2013.]

"Killer" app: partial equivalence checking (PEC) of circuits



source: [Finkbeiner, Tentrup. 2014.]

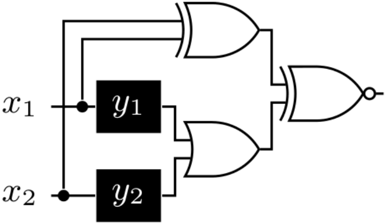
Expansion-based solver:

- expands DQBF to QBF (or even to SAT)
- not publicly available

1st "killer" application

[Gitina, Reimer, Sauer, Wimmer, Scholl, Becker. *Equivalence checking of partial designs using dependency quantified Boolean formulae*. ICCD, 2013.]

"Killer" app: partial equivalence checking (PEC) of circuits



source: [Finkbeiner, Tentrup. 2014.]

Expansion-based solver:

- expands DQBF to QBF (or even to SAT)
- not publicly available

1st publicly available solver

[Finkbeiner, Tentrup. *Fast DQBF Refutation*. SAT, 2014.]
Similar to BMC. Given a bound $k \geq 1$,

- Use k *copies* of all variables and the matrix
- Ackermann constraints as a *guard*:

$$consistent(e, k) := \bigwedge_{1 \leq i,j \leq k} \Big(\bigwedge_{u \in deps_e} u^i = u^j \;\Rightarrow\; e^i = e^j \Big)$$

- Solve the QBF

$$\exists u_1^1, \ldots, u_m^k \; \forall e_1^1, \ldots, e_n^k \; .$$
$$consistent(e_1, k) \wedge \cdots \wedge consistent(e_n, k) \;\Rightarrow\; \bigvee_{1 \leq i \leq k} \neg \phi^k$$

In practice, it can solve only *UNSAT* problems.

1st publicly available "complete" solver – iDQ

[Fröhlich, Kovásznai, Biere. iDQ*: Instantiation-Based* DQBF *Solving*. POS, 2014.]

Adapts and extends the ***Inst-Gen*** approach to DQBF.

Inst-Gen:

- The solving approach for EPR logic
  - The $\exists^\star\forall^\star.\phi$ fragment of 1st-order logic
  - Has the same complexity as DQBF
- The core of iProver, the most successful EPR-solver
- A CEGAR loop generates clause instances by unification

Adaptations: e.g.

- Takes advantage of Boolean domain: uses bit-masks to represents clause instances
- Bit-mask operations for unification, new instances, redundancy check
- VSIDS heuristics

1st publicly available "complete" solver – $\textsc{iDQ}$

[Fröhlich, Kovásznai, Biere. $\textsc{iDQ}$*: Instantiation-Based* DQBF *Solving*. POS, 2014.]

Adapts and extends the ***Inst-Gen*** approach to DQBF.

Inst-Gen:

- The solving approach for EPR logic
  - The $\exists^\star\forall^\star.\phi$ fragment of 1st-order logic
  - Has the same complexity as DQBF
- The core of iProver, the most successful EPR-solver
- A CEGAR loop generates clause instances by unification

Adaptations: e.g.

- Takes advantage of Boolean domain: uses bit-masks to represents clause instances
- Bit-mask operations for unification, new instances, redundancy check
- VSIDS heuristics

1st publicly available "complete" solver – iDQ

DQBF PEC benchmarks

	#(sat/uns)	TO	time	#(sat/uns)	TO	time
	bitcell_16_2			bitcell_16_6		
DQBF2QBF	98 (0/98)	2	18.6	97 (0/97)	3	27.8
iDQ	88 (2/86)	12	128.1	22 (0/22)	78	735.9
iDQ$_{\text{vsids}}$	97 (2/95)	3	39.2	36 (0/36)	64	592.0
iProver	82 (0/82)	18	248.6	7 (0/7)	93	851.7
	adder_3_2			adder_3_6		
DQBF2QBF	94 (0/94)	6	54.8	74 (0/74)	26	234.6
iDQ	82 (1/81)	18	246.8	11 (0/11)	89	841.4
iDQ$_{\text{vsids}}$	43 (0/43)	57	546.3	6 (0/6)	94	863.9
iProver	86 (1/85)	14	221.6	5 (0/5)	95	876.9
	pec_xor2			pec_xor4		
DQBF2QBF	49 (0/49)	51	459.4	99 (0/99)	1	10.6
iDQ	100 (51/49)		.5	100 (1/99)		3.3
iDQ$_{\text{vsids}}$	100 (51/49)		.5	100 (1/99)		2.2
iProver	100 (51/49)		.5	100 (1/99)		2.8

TO = timeout

A new solver – HQS

[Gitina, Wimmer, Reimer, Sauer, Scholl, Becker. *Solving DQBF Through Quantifier Elimination*. DATE, 2015.]

An improved expansion-based solver:

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

$$\forall u_1, u_2 \exists e(u_1) \,.\, \phi \;\longrightarrow\; \forall u_2 \exists e, e' \,.\, \phi[0/u_1] \wedge \phi[1/u_2][e'/e]$$

- Eliminates the *minimum set* of variables that cause non-linear dependencies
  - Expressed as a partial MaxSAT problem
- Uses AIGs to detect units and pure literals
- Publicly available?

A new solver – HQS

DQBF PEC benchmarks

	#(sat/uns)	TO/MO	time	#(sat/uns)	TO/MO	time
	adder			bitcell		
HQS	300 (42/258)	0/0	9.7	300 (7/293)	0/0	11.3
iDQ	216 (3/213)	84/0	89828	190 (2/188)	110/0	78107
	lookahead			pec_xor		
HQS	300 (10/290)	0/0	23.2	200 (24/176)	0/0	33.6
iDQ	273 (4/269)	27/0	39540	200 (24/176)	0/0	181.6
	z4			comp		
HQS	240 (72/168)	0/0	4.9	155 (39/116)	9/76	17.8
iDQ	111 (8/103)	129/0	41626	25 (0/25)	180/35	11.6
	C432					
HQS	60 (19/41)	0/180	1333			
iDQ	20 (0/20)	85/135	0.2			

TO = timeout
MO = memory out

Preprocessing for DQBF

When experimenting with iDQ, we tried out simple preprocessing techniques:

- Dependency set reduction $\Rightarrow$ did not pay off
  - by using the standard dependency scheme (such as in DepQBF, by Lonsing);
  - by using resolution-path dependency scheme (by Slivovsky, Szeider)
- Blocked clause elimination (BCE)

Preprocessing with iDQ

DQBF PEC benchmarks

	#(sat/uns)	TO	time	#(sat/uns)	TO	time
	bitcell_16_2			bitcell_16_6		
iDQ	88 (2/86)	12	128.1	22 (0/22)	78	735.9
iDQ_{BCE}	100 (2/98)		.7	95 (0/95)	5	49.5
$\text{iDQ}_{\text{vsids}}$	97 (2/95)	3	39.2	36 (0/36)	64	592.0
$\text{iDQ}_{\text{vsids+BCE}}$	100 (2/98)		.7	85 (0/85)	15	185.6
	lookahead_16_2			lookahead_16_6		
iDQ	82 (1/81)	18	246.8	11 (0/11)	89	841.4
iDQ_{BCE}	100 (3/97)		.7	87 (1/86)	13	132.4
$\text{iDQ}_{\text{vsids}}$	43 (0/43)	57	546.3	6 (0/6)	94	863.9
$\text{iDQ}_{\text{vsids+BCE}}$	100 (3/97)		.9	6 (0/6)	94	853.9

Preprocessing for DQBF

There are some rumors about a SAT'15 paper on DQBF preprocessing.
It is said to be great... :)

Conclusion

- DQBF solving is getting more and more serious
    - Complex and sophisticated solving approaches: e.g., CEGAR, QBF solver back-end, MaxSAT, clever heuristics, etc.
- Preprocessing in on the way...
- *Industrial* DQBF instances should appear soon
- Any other "natural" application for DQBF?