DQBDD: An Efficient BDD-Based DQBF Solver

Juraj Síč
Jan Strejček

What is DQBF

- DQBF = **dependency** quantified Boolean formula
- Like QBF + **explicit** dependencies between variables
- QBF: $\forall x_1 \exists y_1 \forall x_2 \exists y_2 . (x_1 \wedge x_2) \Leftrightarrow (y_1 \Leftrightarrow y_2)$
  - y_1 depends only on x_1
  - y_2 depends on both x_1 and x_2
  - satisfiable
- DQBF: $\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) . (x_1 \wedge x_2) \Leftrightarrow (y_1 \Leftrightarrow y_2)$
  - y_1 depends only on x_1
  - y_2 depends only on x_2
  - not satisfiable

Approach

- Main solving method: **quantifier elimination**
- Uses **binary decision diagrams** (BDD)
  - for propositional subformulas
- Enhanced by:
  - **quantifier localisation**
  - preprocessing: **HQSpre**

Quantifier Elimination

- Iteratively eliminate variables until true/false remains
- For QBF:

$$\forall x.\psi \approx \psi[1/x] \wedge \psi[0/x]$$
$$\exists y.\psi \approx \psi[1/y] \vee \psi[0/y]$$

- Same for DQBF but:
  - universal QE introduces new variables
  - existential QE is not always applicable

Universal QE

$$\forall x.\psi \approx \psi_1[1/x] \wedge \psi_2[0/x]$$

- ψ_1 is ψ but x is removed from dependency sets of existential variables
- ψ_2 is ψ_1 but each existential variable y depending on x is replaced by a new copy y'
- Example:

$$\forall \boldsymbol{x} \exists y_1(\emptyset) \exists y_2(\boldsymbol{x}).(y_1 \wedge y_2) \vee \boldsymbol{x}$$
$$\approx \exists y_1(\emptyset) \exists y_2(\emptyset) \exists y_2'(\emptyset).((y_1 \wedge y_2) \vee \mathbf{1}) \wedge ((y_1 \wedge y_2') \vee \mathbf{0})$$

Existential QE

$$\exists y(D_y).\psi \approx \psi[1/y] \vee \psi[0/y]$$

- ψ must be quantifier free
- Each variable in ψ must be either:
  - free variable
  - universal variable from D_y
  - existential variable y' with $D_{y'} \subseteq D_y$
- Example:

$$\forall x \exists y_1(\emptyset) \exists \boldsymbol{y_2}(x).(y_1 \wedge \boldsymbol{y_2}) \vee x$$
$$\approx \forall x \exists y_1(\emptyset).((y_1 \wedge \mathbf{1}) \vee x) \vee ((y_1 \wedge \mathbf{0}) \vee x)$$

Algorithm

1. Input formats
   - DQDIMACS – prenex CNF
   - DQCIR – prenex non-CNF

```
p cnf 6 2
a 1 2 0
e 3 0
a 4 0
e 5 0
d 6 1 4 0
-1 6 -3 5 0
-2 -4 5 0
```

a DQDIMACS

```
#QCIR-G14 10
forall(1, 2)
exists(3)
forall(4)
exists(5)
depend(6, 1, 4)
output(7)
8 = and(-6, 3)
9 = or(-1, -8, 5)
10 = or(-2, -4, 5)
7 = and(9, 10)
```

b DQCIR

Algorithm

1. Input formats
   - DQDIMACS – prenex CNF
   - DQCIR – prenex non-CNF
2. Apply preprocessing – HQSpre (only DQDIMACS)

```
p cnf 6 2
a 1 2 0
e 3 0
a 4 0
e 5 0
d 6 1 4 0
-1 6 -3 5 0
-2 -4 5 0
```

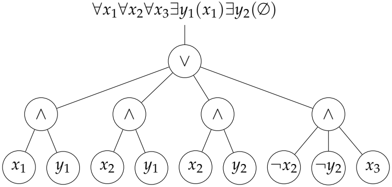
a DQDIMACS

```
#QCIR-G14 10
forall(1, 2)
exists(3)
forall(4)
exists(5)
depend(6, 1, 4)
output(7)
8 = and(-6, 3)
9 = or(-1, -8, 5)
10 = or(-2, -4, 5)
7 = and(9, 10)
```

b DQCIR

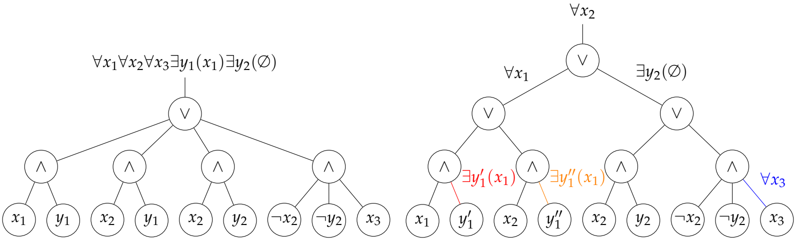
Algorithm

1. Input formats
   - DQDIMACS – prenex CNF
   - DQCIR – prenex non-CNF
2. Apply preprocessing – HQSpre (only DQDIMACS)
3. Transform into quantifier tree



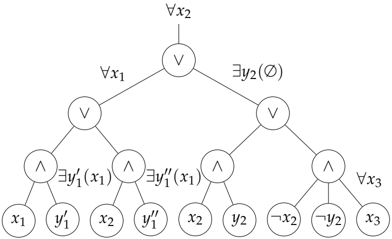
Algorithm

1. Input formats
   - DQDIMACS – prenex CNF
   - DQCIR – prenex non-CNF
2. Apply preprocessing – HQSpre (only DQDIMACS)
3. Transform into quantifier tree and push quantifiers inside



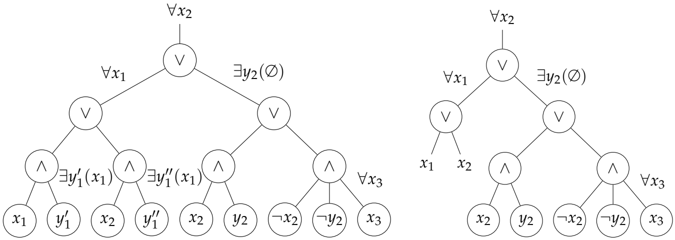
Algorithm

4. Iteratively transform to BDD while eliminating quantifiers



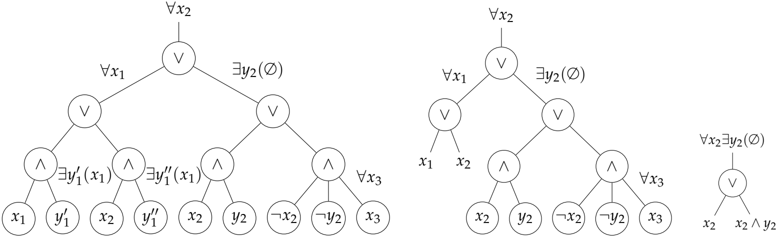
Algorithm

4. Iteratively transform to BDD while eliminating quantifiers



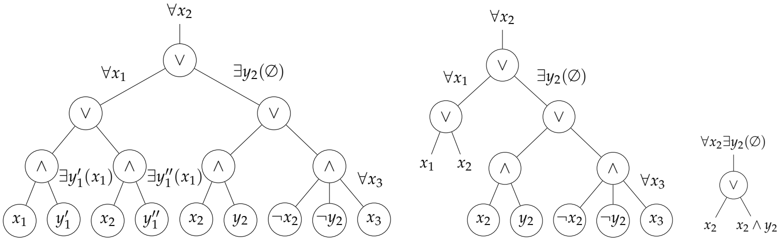
Algorithm

4. Iteratively transform to BDD while eliminating quantifiers



Algorithm

4. Iteratively transform to BDD while eliminating quantifiers
5. Return SAT/UNSAT if the resulting BDD represents true/false



Strategies and Heuristics

- Quantifier elimination strategies:
  - **none**
  - **simple** (default)
  - **all**
- Universal quantifier elimination order heuristics:
  - **at the beginning** (default)
  - **current lowest**
  - **vars in conjuncts**

Implementation and Usage

- Implemented in C++
- BDD library CUDD
- `https://github.com/jurajsic/DQBDD`
  - source code
  - binaries for Linux and Mac
- Demonstration

Comparison With Other Solvers

- DQBF solvers: **dCAQE**, **HQS**, **iDQ**, **iProver**
- Preprocessing (**HQSpre**) used for all solvers
- Benchmarks:
  - 3277 **partial equivalence checking** instances } next slides
  - 404 **controller synthesis problem** instances } next slides
  - 22 **SAT** instances encoded as DQBF – worse than others
- **Winner** of the DQBF track of QBF Evaluation 2020

Comparison With Other Solvers – PEC

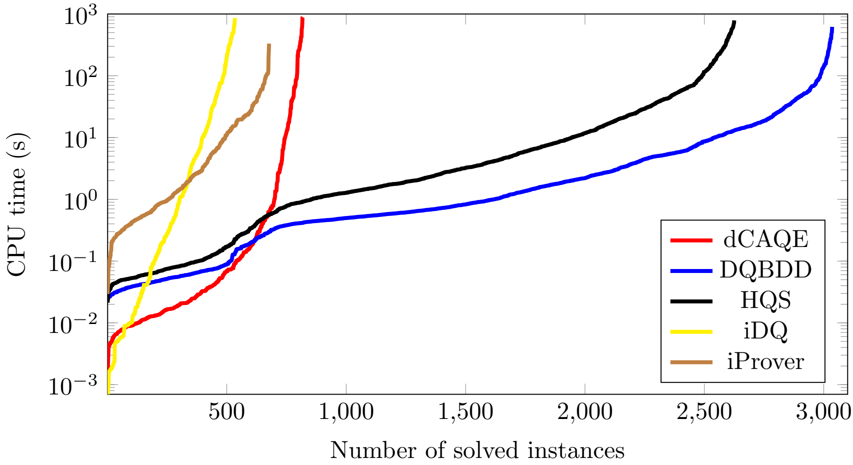


Figure: Cactus plot comparing DQBF solvers for instances of partial equivalence checking

Comparison With Other Solvers – CSP

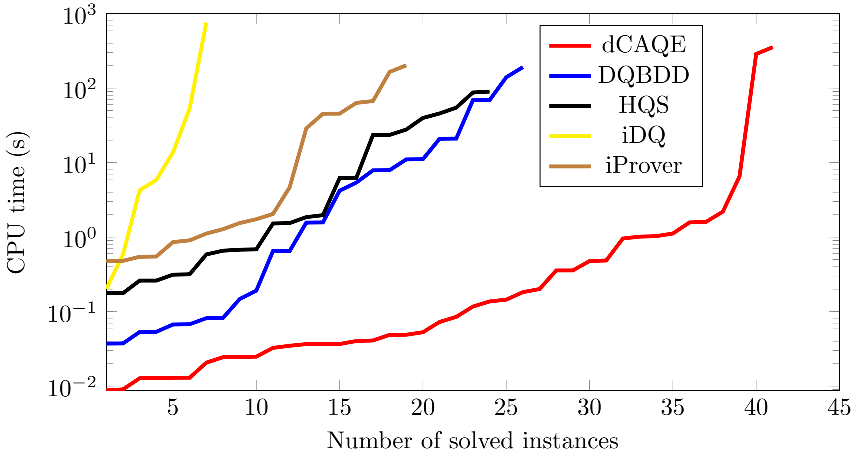


Figure: Cactus plot comparing DQBF solvers for instances of controller synthesis problem

Thank You for Your Attention!