Integrating Dependency Schemes in Search-Based QBF Solvers

Florian Lonsing and Armin Biere

Institute for Formal Models and Verification (FMV)
Johannes Kepler University, Linz, Austria
`http://fmv.jku.at`

SAT'10
July 11 - July 14, 2010
Edinburgh, Scotland, United Kingdom

JOHANNES KEPLER UNIVERSITY LINZ | JKU

TNF

Overview

Search-based QBF Solving for Prenex-CNF (PCNF): QDPLL

- Classical approach relies on linear quantifier prefix: $Q_1 Q_2 \ldots Q_n.\ \phi$.
- Analyzing variable dependencies: can prefix order be relaxed?

Example (Dependencies: quantifier ordering matters)

- $\forall x \exists y.\ (x = y)$ is satisfiable: value of y *depends* on value of x.
- $\exists y \forall x.\ (x = y)$ is unsatisfiable: value of y is fixed for all values of x.

This Talk:

- Making QDPLL aware of dependencies.
- Identifying independence.
- Search-based QBF solving + dependency analysis.
- Implementation: DepQBF.

Solver	*Score*
DepQBF	**2896.68**
DepQBF-pre	2508.96
aqme-10	2467.96
qmaiga	2117.55
AIGSolve	2037.22
quantor-3.1	1235.14
struqs-10	947.83
nenofex-qbfeval10	829.11

QBFEVAL'10: score-based ranking.

Dependency Representation

[SS09, Bie04, BB07, Ben05]

Dependency Schemes: $D \subseteq (V_\exists \times V_\forall) \cup (V_\forall \times V_\exists)$.

- General framework for expressing (in)dependence in PCNFs.
- $(x, y) \notin D$: y independent from x.
- $(x, y) \in D$: *conservatively* regard y as depending on x.

D **as Directed-Acyclic Graph (Dependency-DAG):**

- Edges $x \rightarrow y$ iff $(x, y) \in D$.

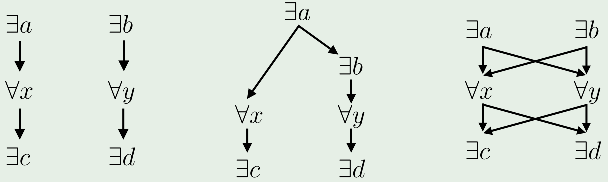
Constructing Dependency Schemes

Syntactic Approaches:

- *Trivial dependency scheme* D^{triv} (given prefix).
- *Quantifier trees* D^{tree}.
- *Standard dependency scheme* D^{std}.
- Theory: $D^{\text{std}} \subseteq D^{\text{tree}} \subseteq D^{\text{triv}}$.

Example

$$\exists a, b \forall x, y \exists c, d.\ (a \vee x \vee c) \wedge (a \vee b) \wedge (b \vee d) \wedge (y \vee d).$$



D^{std} D^{tree} D^{triv}

Goal: dependency-DAG for D^{std} in QDPLL ⇒ expecting more freedom.

[CGS98, GNT07, LB09, LB10]

Related Work:

- Classical result: description of QDPLL with D^{triv}.
- First generalization: QDPLL with D^{tree}.

Our Results:

- Description of QDPLL for *arbitrary* dependency schemes.
- Solver DepQBF: implementation of QDPLL with D^{std}.
- Previous work: compact dependency-DAG representation for D^{std}.
- Experimental evaluation.

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

Parts to be Generalized from D^{triv} to Arbitrary $D \subseteq D^{\text{triv}}$:

- Unit literal detection: expecting more units.
- Learning: expecting shorter constraints and "enabled" resolution steps.
- Stop criterion/asserting level: expecting longer backjumps.
- Selection of decision variables: expecting more freedom.

Focus: Decision Making

Decisions and Dependency-Order:

- Out-of-order decisions: generally unsound $\Rightarrow$ must branch in "*D*-order".

Example

$\exists a \forall x, y \exists b.\ \phi$: branching on b possible by D^{triv} only if x, y assigned.

Decision Candidates (DC):

- Def.: unassigned variables x where all $y \in \overline{D}(x)$ are assigned.
- Candidate has all "preconditions" wrt. D assigned, i.e. is *enabled*.

Example

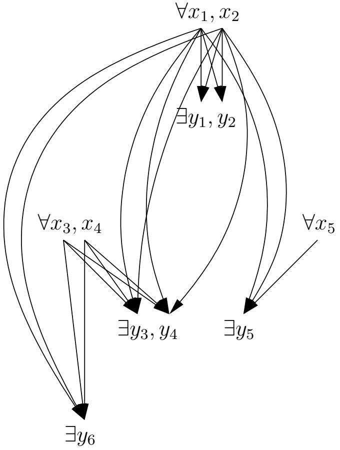
$\exists a \forall x, y \exists b.\ \phi$: assigning a enables x and y by D^{triv}.

Lazy DC-Maintenance:

- DCs needed exactly before making a decision and not e.g. during BCP.
- Defer updating set of DCs as long as possible.
- Exploit DAG structure for incremental updates.

Decision Making: Compact Dependency-DAG for D^{std}

[LB09]

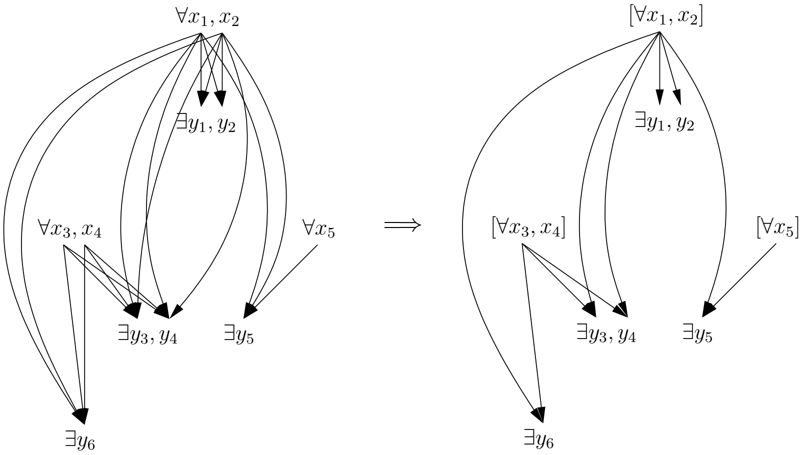


$\Longrightarrow$

- For simplicity: ignoring dependencies of the form $\exists x \rightarrow \forall y$.
- DAG: edges $x \rightarrow y$ iff $(x, y) \in D$.
- Construct dependency-DAG over equivalence classes of variables.

Decision Making: Compact Dependency-DAG for D^{std}

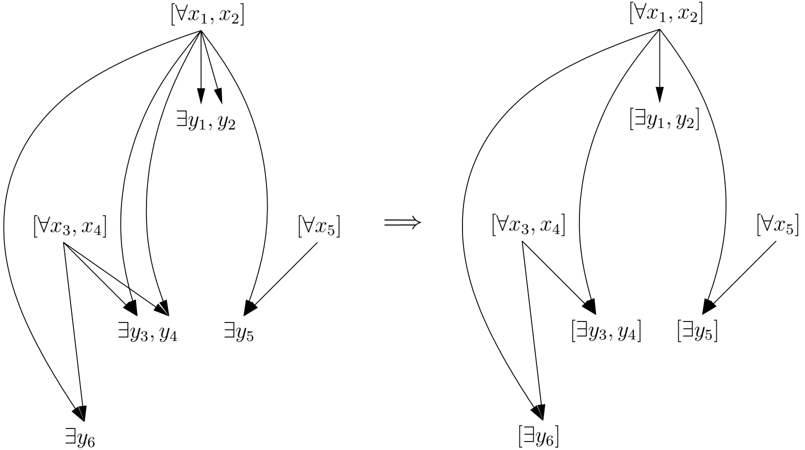
[LB09]



- Merge $\forall x$ and $\forall y$ if have same *outgoing* pointers, i.e. $D(x) = D(y)$.

Decision Making: Compact Dependency-DAG for D^{std}

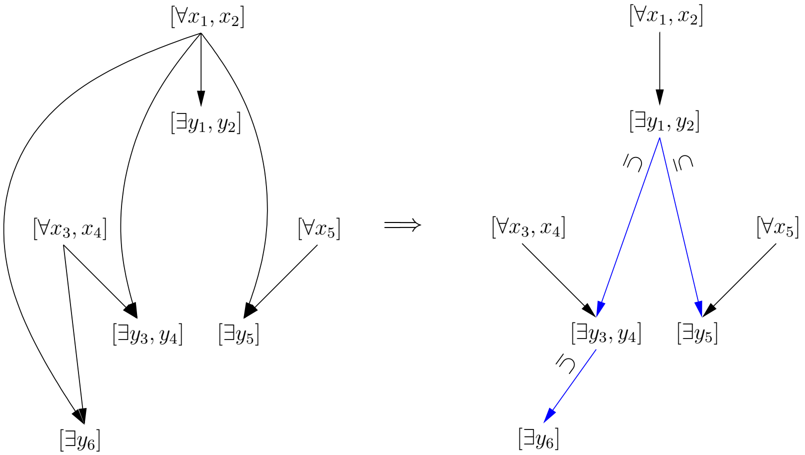
[LB09]



- Merge $\exists x$ and $\exists y$ if have same *incoming* pointers, i.e. $\overline{D}(x) = \overline{D}(y)$.

Decision Making: Compact Dependency-DAG for D^{std}

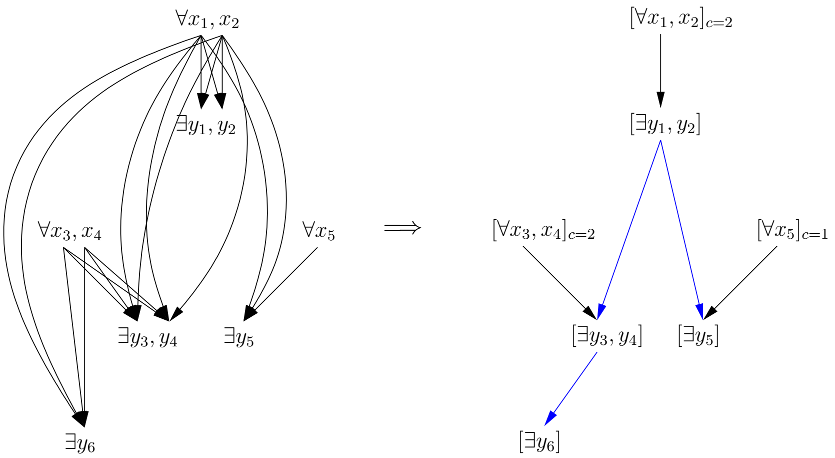
[LB09]



- Add non-transitive edges $[\exists x] \longrightarrow [\exists y]$ if $\overline{D}(x) \subseteq \overline{D}(y)$.
- Delete redundant $\forall \rightarrow \exists$ edges.

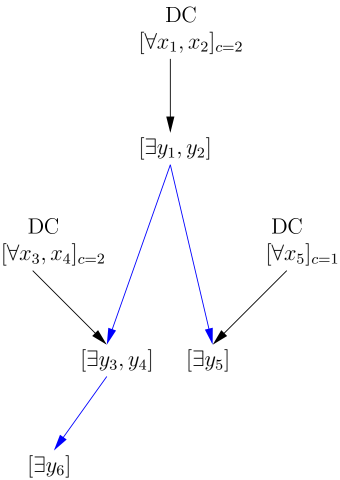
Decision Making: Compact Dependency-DAG for D^{std}

[LB09]

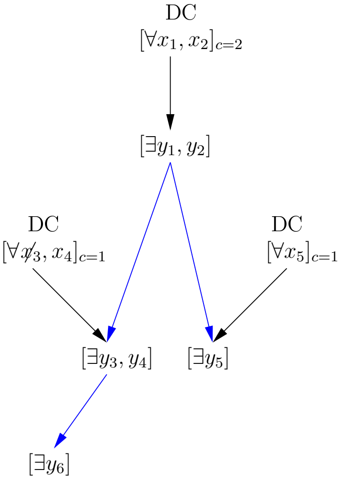


- Only fully assigned *classes* possibly enable new candidates.
- Counters c in $\forall$-classes: number of unassigned variables in class.
- Relevant cases: $c = 1 \rightarrow c = 0$ and $c = 0 \rightarrow c = 1$.

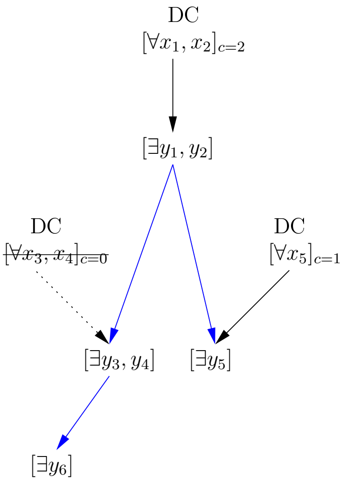
- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.



- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.

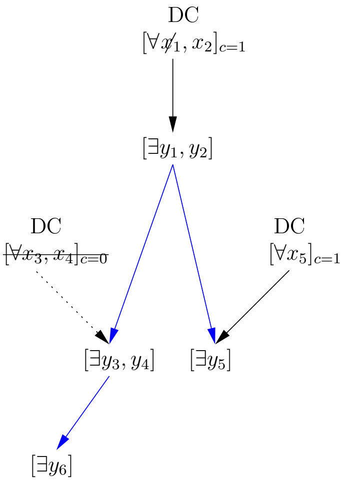


- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.

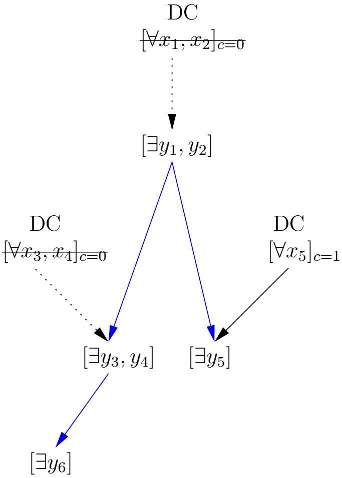


Decision Making: Example

- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.

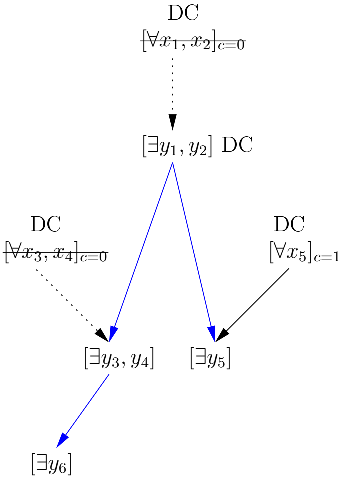


- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.

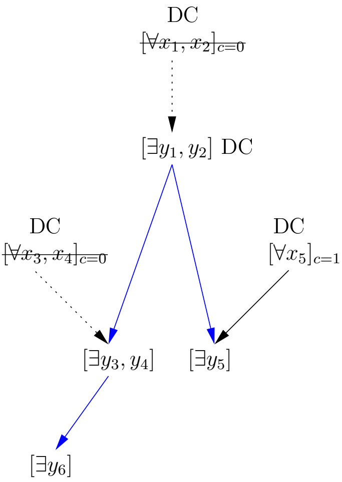


Decision Making: Example

- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.

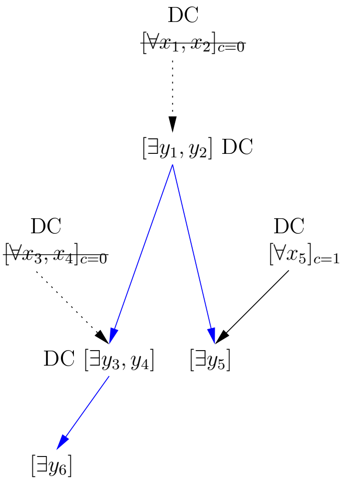


- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.



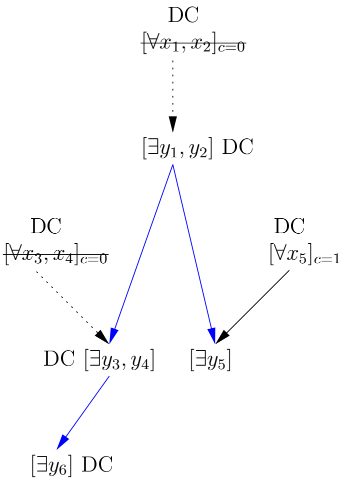
Decision Making: Example

- Assign x_3: $c = 2 \to c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \to c = 0$.
  - Follow $[\forall] \to [\exists] \to^* [\exists]$ edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \to c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \to c = 0$.
  - Follow $[\forall] \to [\exists] \to^* [\exists]$ edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned $[\forall]$-refs.
  - $[\exists y_5]$ not DC, has unassigned $[\forall]$-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned $[\forall]$-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned $[\forall]$-refs.

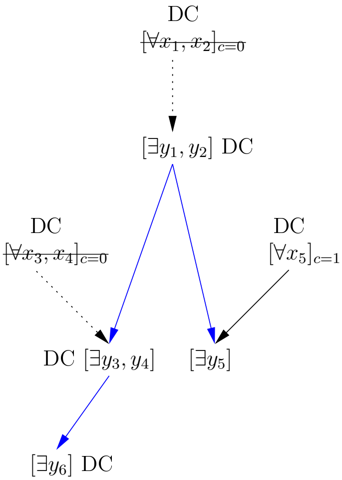


Decision Making: Example

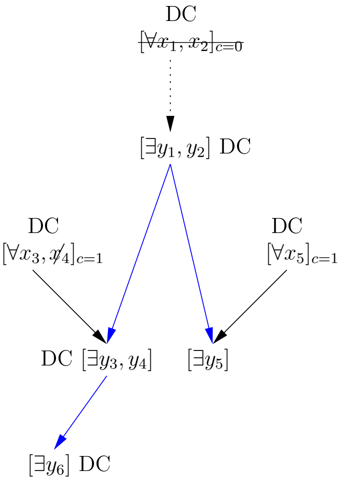
- Assign x_3: $c = 2 \rightarrow c = 1$.
  - No change, x_4 still unassigned.
- Assign x_4: $c = 1 \rightarrow c = 0$.
  - Follow [∀] → [∃] →* [∃] edges.
  - Parent $[\exists y_1, y_2]$ of $[\exists y_3, y_4]$ not DC.
  - $[\exists y_1, y_2]$ has unassigned [∀]-refs.
  - $[\exists y_3, y_4]$ also ref'd by $[\forall x_1, x_2]$.
  - $\Rightarrow [\exists y_3, y_4]$ not DC.
- Assign x_1: $c = 2 \rightarrow c = 1$.
  - No change, x_2 still unassigned.
- Assign x_2: $c = 1 \rightarrow c = 0$.
  - Follow [∀] → [∃] →* [∃] edges.
  - $[\exists y_1, y_2]$ new DC: no unassigned [∀]-refs.
  - $[\exists y_5]$ not DC, has unassigned [∀]-refs.
  - $[\exists y_3, y_4]$ new DC: parent $[\exists y_1, y_2]$ DC and no unassigned [∀]-refs.
  - $[\exists y_6]$ new DC: parent $[\exists y_3, y_4]$ DC and no unassigned [∀]-refs.



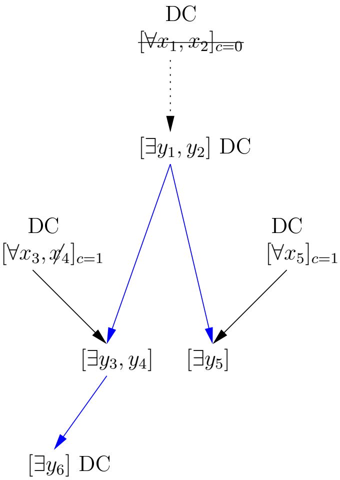
- Unassign x_3: $c = 0 \rightarrow c = 1$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Disabling $[\exists y_3, y_4]$, ref'd by $[\forall x_3, x_4]$.
  - Disabling $[\exists y_6]$, ref'd by $[\exists y_3, y_4]$.
- Unassign x_4: $c = 1 \rightarrow c = 2$.
  - No additional work done.



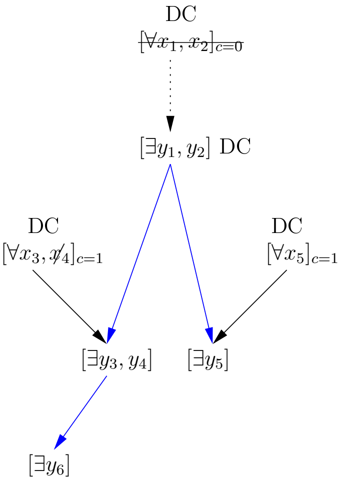
- Unassign x_3: $c = 0 \rightarrow c = 1$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Disabling $[\exists y_3, y_4]$, ref'd by $[\forall x_3, x_4]$.
  - Disabling $[\exists y_6]$, ref'd by $[\exists y_3, y_4]$.
- Unassign x_4: $c = 1 \rightarrow c = 2$.
  - No additional work done.



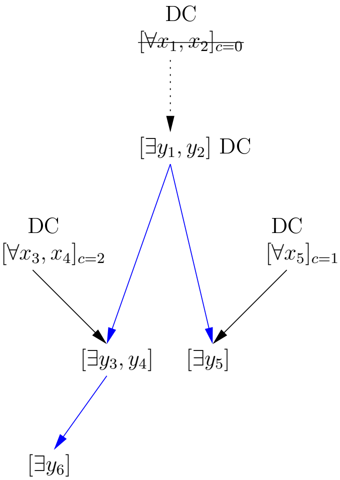
- Unassign x_3: $c = 0 \rightarrow c = 1$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Disabling $[\exists y_3, y_4]$, ref'd by $[\forall x_3, x_4]$.
  - Disabling $[\exists y_6]$, ref'd by $[\exists y_3, y_4]$.
- Unassign x_4: $c = 1 \rightarrow c = 2$.
  - No additional work done.



- Unassign x_3: $c = 0 \rightarrow c = 1$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Disabling $[\exists y_3, y_4]$, ref'd by $[\forall x_3, x_4]$.
  - Disabling $[\exists y_6]$, ref'd by $[\exists y_3, y_4]$.
- Unassign x_4: $c = 1 \rightarrow c = 2$.
  - No additional work done.



- Unassign x_3: $c = 0 \rightarrow c = 1$.
  - Follow $[\forall] \rightarrow [\exists] \rightarrow^* [\exists]$ edges.
  - Disabling $[\exists y_3, y_4]$, ref'd by $[\forall x_3, x_4]$.
  - Disabling $[\exists y_6]$, ref'd by $[\exists y_3, y_4]$.
- Unassign x_4: $c = 1 \rightarrow c = 2$.
  - No additional work done.



QBFEVAL'08 (3326 formulae)					
	D^{triv}	D^{tree}	D^{std}	*QuBE6.6-np*	*QuBE6.6*
solved	1223	1221	**1252**	1106	**2277**
avg. time	579.94	580.64	**572.31**	608.97	**302.49**
QBFEVAL'07 (1136 formulae)					
solved	533	548	**567**	458	**734**
avg. time	497.12	484.69	**469.97**	549.29	**348.05**

Table: Comparison of DepQBF with $D^{\text{std}} \subseteq D^{\text{tree}} \subseteq D^{\text{triv}}$ and QuBE6.6.

DepQBF:

- QDPLL for PCNF with with clause- and cube-learning.
- Dependency-DAG for D^{std} (primary), and D^{tree}, D^{triv} (experimentally).
- No preprocessing.
- D^{std} pays off despite DAG-overhead.
- More solved instances in less time.
- But: preprocessing is important.
- Reference: QuBE6.6 with(out) preprocessing (*QuBE6.6-np*) [GNT01].

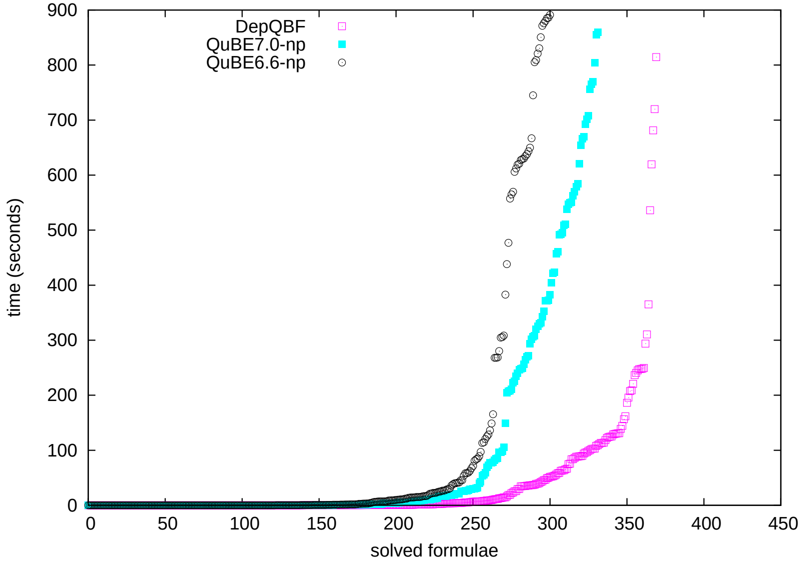
QBFEVAL'10 main track (568 formulae)						
	Solved+Unsolved		*Solved SAT*		*Solved UNSAT*	
	solved	*avg.time*	*solved*	*avg.time*	*solved*	*avg.time*
QuBE7.0-pre⇒DepQBF	**424**	**254.23**	**197**	**48.17**	**227**	**23.42**
QuBE7	414	310.29	187	130.52	227	58.33
QuBE6.6	387	341.91	168	98.97	219	67.03
without preprocessing						
DepQBF	**370**	**337.10**	**165**	**54.58**	**205**	**20.82**
QuBE7.0-np	332	425.44	135	147.71	197	47.27
QuBE6.6-np	301	468.51	113	136.48	188	55.27

Table: Comparison of DepQBF with D^{std} and state-of-the-art QBF solvers. Ranking by number of solved formulae. Statistics include time for preprocessing.

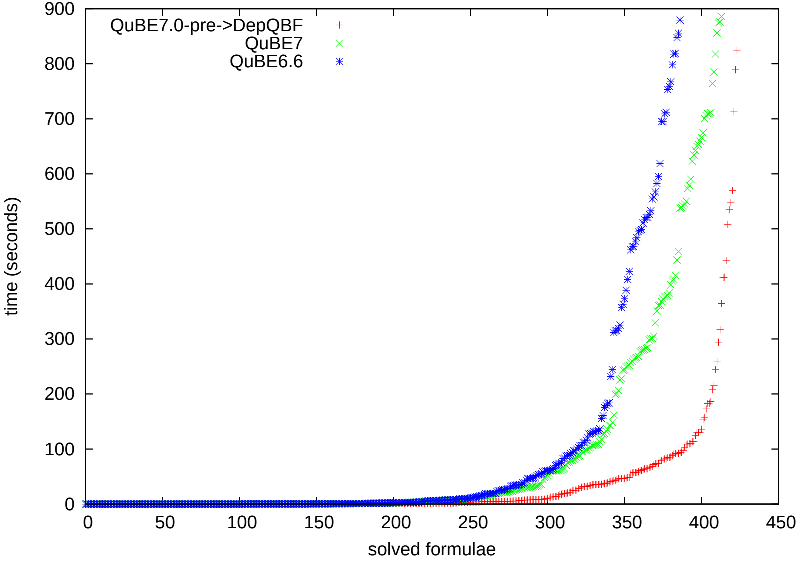
Preprocessing:

- DepQBF with D^{std}: best both with and without preprocessing.
- QuBE7.0-pre: preprocessor integrated in QuBE7.0 [GMN10].

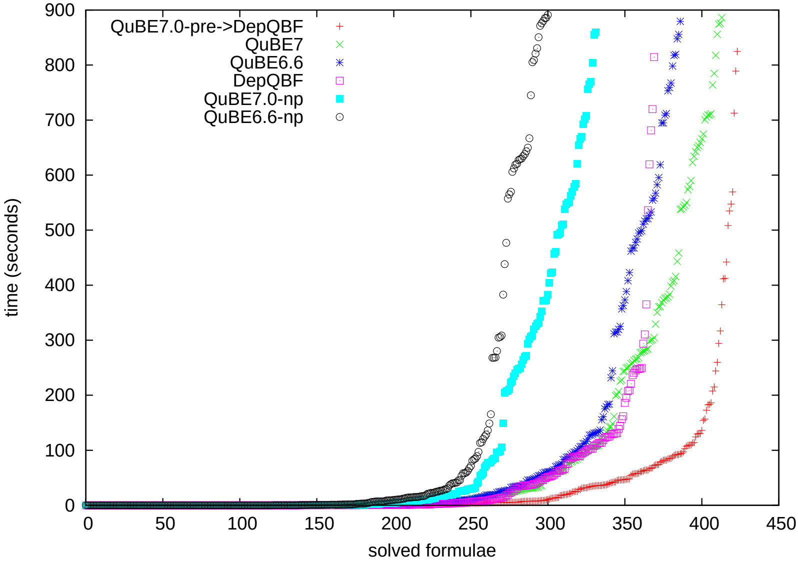
Experimental Results 3/5: QBFEVAL'10



Experimental Results 4/5: QBFEVAL'10



Experimental Results 5/5: QBFEVAL'10



QDPLL with Dependency Schemes:

- $D \subseteq D^{\mathrm{triv}}$ relaxes prefix order to allow more freedom in QDPLL.

Implementation: QDPLL-based solver DepQBF.

- Compact dependency-DAG for D^{std} over equivalence classes.
- Top-ranked solver in QBFEVAL'10: D^{std} pays off despite DAG-overhead.
- See also *Pragmatics of SAT 2010 (POS'10)* workshop:
  "DepQBF: A Dependency-Aware QBF Solver (System Description)".

Future Work:

- Preprocessing,. . .

DepQBF 0.1 is open source: `http://fmv.jku.at/depqbf/`

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

Boolean Constraint Propagation (BCP):

- Assigning unit and pure literals.
- Augmented CNF: $\phi := \phi_{OCL} \wedge \phi_{LCL} \vee \phi_{OCU}$.
- Original clauses ϕ_{OCL}, learnt clauses ϕ_{LCL} and learnt cubes ϕ_{LCU}.

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

Decision Making:

- BCP saturated without detecting conflict/solution.
- Select and assign one *decision candidate*.
- Candidates: according to dependency scheme and partial assignment.

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

Result Analysis:

- BCP detected conflict/solution.
- Conflict: empty clause.
- Solution: satisfying assignment or satisfied learnt cube.

[APPENDIX] – QDPLL 5/6

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

Constraint Learning:

- Init. from conflict: empty clause.
- Init. from solution: sat. cube or new cube from satisfying assignment.
- Resolution/consensus: antecedents of units in current clause/cube.
- First-UIP: generalized stop criterion.

[GNT02, Let02, ZM02, CGS98, GNT06, BKF95]

```
State qdpll ()
  while (true)
    State s = bcp ();
    if (s == UNDEF)
      // Make decision.
      v = select_dec_var ();
      assign_dec_var (v);
    else
      // Conflict or solution.
      // s == UNSAT or s == SAT.
      btlevel = analyze_leaf (s);
      if (btlevel == INVALID)
        return s;
      else
        backtrack (btlevel);
```

```
DecLevel analyze_leaf (State s)
  R = get_initial_reason (s);
  // s == UNSAT: 'R' is empty clause.
  // s == SAT: 'R' is sat. cube...
  // ..or new cube from assignment.
  while (!stop_res (R))
    p = get_pivot (R);
    A = get_antecedent (p);
    R = constraint_res (R, p, A);
  add_to_formula (R);
  assign_forced_lit (R);
  return get_asserting_level (R);
```

Figure: QDPLL with conflict-driven clause and solution-driven cube learning.

Backtrack:

- Assumption: learning always produces asserting constraints.
- Backtrack to asserting level.

[CGS98, GNT02, ZM02, GNT07]

Given dependency scheme D for PCNF. Write $x \prec v$ if $(x, y) \in D$.

Definition (Unit Clause Rule)

A clause C is *unit* iff (Dual definition for cubes.)

- no literal $l \in C$ is assigned true,
- exactly one existential literal $l_e \in L_\exists(C)$ is unassigned,
- for all unassigned universal literals $l_u \in L_\forall(C)$: $l_u \not\prec l_e$.

Example: $\exists x \forall a \exists y,z.\ \phi' \wedge (x \vee a \vee y \vee z)$.

Assign $\overline{x}, \overline{y}$: $\exists x \forall a \exists y,z.\ \phi' \wedge (x \vee a \vee y \vee z)$.

Given D^{triv} from prefix: $(x \vee a \vee y \vee z)$ *not* unit since $a \prec z$ (because $\forall a$ before $\exists z$).

Given $D \subseteq D^{\text{triv}}$ where $a \not\prec z$: $(x \vee a \vee y \vee z)$ unit.

Practical Effects:

- Expecting more units when using $D \subseteq D^{\text{triv}}$.
- Combining two-literal watching with dependency checking.

[BKF95, GNT02, ZM02]

Constraint Reduction: universal/existential reduction of clauses/cubes.

Definition (Universal Reduction of Clauses)

A universal literal $l_u \in L_\forall(C)$ can be deleted from a clause C iff

- there is no $l_e \in L_\exists(C)$ with $l_u \prec l_e$.
- The result of saturated universal reduction is denoted by $CR(C)$.

(Dual definition of existential reduction for cubes.)

Example: $\exists x \forall a \exists y.\ \phi' \wedge (x \vee a \vee y)$.

Given D^{triv} from prefix: a is irreducible in $(x \vee a \vee y)$ since $a \prec y$.

Given $D \subseteq D^{\text{triv}}$ where $a \not\prec y$: a is reducible in $(x \vee a \vee y)$, yielding $(x \vee y)$.

Practical Effects:

- Expecting shorter learnt constraints when using $D \subseteq D^{\text{triv}}$.
- Combining constraint reduction with dependency checks.

[BKF95, GNT02, ZM02, Let02, GNT06]

Constraint Resolution: Q-resolution/consensus of clauses/cubes.

Definition (Q-resolution for Clauses)

Clarifies Def. 7 in paper.

Let C_1, C_2 be clauses with $v \in L_\exists(C_1), \overline{v} \in L_\exists(C_2)$.

1. $C := (CR(C_1) \cup CR(C_2)) \setminus \{v, \overline{v}\}$.
2. If C contains complementary literals then no resolvent exists.
3. Otherwise, resolvent $C' := CR(C)$ of C_1 and C_2 on v: $\{C_1, C_2\} \vdash_v C'$.

(Dual definition of consensus for cubes.)

Example: $\exists x \forall a \exists y,z.\ \phi' \wedge \overbrace{(x \vee a \vee y \vee z)}^{C_1} \wedge \overbrace{(x \vee a \vee y \vee \overline{z})}^{C_2} \wedge \overbrace{(x \vee \overline{a} \vee \overline{y} \vee z)}^{C_3}$.

Given D^{triv} from prefix: $\{C_1, C_2\} \vdash_z (x \vee a \vee y)$, but $\{(x \vee a \vee y), C_3\} \not\vdash_y$.

Given $D \subseteq D^{\text{triv}}$ where $a \not\prec y$: $\{C_1, C_2\} \vdash_z (x \vee y)$, and $\{(x \vee y), C_3\} \vdash_y (x \vee \overline{a} \vee z)$.

Practical Effects:

- Possible reductions of “resolution-blocking” literals when using $D \subseteq D^{\text{triv}}$.

[APPENDIX] – Stop Criterion

[ZM02, GNT06]

Definition (Asserting Clause/Level)

Clarifies Def. 8 from paper. See also function `get_reason_asserting_level` in DepQBF 0.1 source code.

Let R be a resolvent i.e. $\{\ldots\} \vdash^* R$. Let $d := max(\{dl(l) \mid l \in L_\exists(R)\})$. R is *asserting* at $a := max(\{dl(l) < d \mid l \in L_\exists(R)$ or $l \in L_\forall(R)$ with $l \prec d\})$ iff

1. the decision variable at level d is existential,
2. there is exactly one $l \in L_\exists(R)$ with $dl(l) = d$,
3. for all $l_u \in L_\forall(R)$ where $l_u \prec l$: l_u must be assigned false with $dl(l_u) < d$.

(Dual definition for asserting cubes.)

Example: $\ldots \exists x \ldots \forall a \ldots \exists y, z \ldots \ \phi' \wedge (x \vee a \vee y \vee z)$.

Given D^{triv} from prefix: in $(\overset{@1}{x} \vee \overset{@3}{a} \vee \overset{@2}{y} \vee \overset{@4}{z})$, z is unit at level 3.

Given $D \subseteq D^{\text{triv}}$ where $a \not\prec z$: in $(\overset{@1}{x} \vee \overset{@3}{a} \vee \overset{@2}{y} \vee \overset{@4}{z})$, z is unit at level 2.

Practical Effects:

- Possibly longer backjumps when using $D \subseteq D^{\text{triv}}$.

U. Bubeck and H. Kleine Büning.
Bounded Universal Expansion for Preprocessing QBF.
In J. Marques-Silva and K. A. Sakallah, editors, *SAT*, volume 4501 of *LNCS*, pages 244–257. Springer, 2007.

M. Benedetti.
Quantifier Trees for QBFs.
In F. Bacchus and T. Walsh, editors, *SAT*, volume 3569 of *LNCS*, pages 378–385. Springer, 2005.

A. Biere.
Resolve and Expand.
In H. H. Hoos and D. G. Mitchell, editors, *SAT (Selected Papers)*, volume 3542 of *LNCS*, pages 59–70. Springer, 2004.

H. Kleine Büning, M. Karpinski, and A. Flögel.
Resolution for Quantified Boolean Formulas.
Inf. Comput., 117(1):12–18, 1995.

M. Cadoli, A. Giovanardi, and M. Schaerf.
An Algorithm to Evaluate Quantified Boolean Formulae.
In *AAAI/IAAI*, pages 262–267, 1998.

E. Giunchiglia, P. Marin, and M. Narizzano.
sQueezeBF: An Effective Preprocessor for QBFs.

In O. Strichman and S. Szeider, editors, *SAT (accepted for publication)*, LNCS. Springer, 2010.

E. Giunchiglia, M. Narizzano, and A. Tacchella.
QUBE: A System for Deciding Quantified Boolean Formulas Satisfiability.
In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR*, volume 2083 of *LNCS*, pages 364–369. Springer, 2001.

E. Giunchiglia, M. Narizzano, and A. Tacchella.
Learning for Quantified Boolean Logic Satisfiability.
In *AAAI/IAAI*, pages 649–654, 2002.

E. Giunchiglia, M. Narizzano, and A. Tacchella.
Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas.
J. Artif. Intell. Res. (JAIR), 26:371–416, 2006.

E. Giunchiglia, M. Narizzano, and A. Tacchella.
Quantifier Structure in Search-Based Procedures for QBFs.
TCAD, 26(3):497–507, 2007.

F. Lonsing and A. Biere.
A Compact Representation for Syntactic Dependencies in QBFs.
In O. Kullmann, editor, *SAT*, volume 5584 of *LNCS*, pages 398–411. Springer, 2009.

F. Lonsing and A. Biere.
DepQBF: A Dependency-Aware QBF Solver (System Description).
In A. Van Gelder and D. Le Berre, editors, *Pragmatics of SAT Workshop (POS), accepted for publication*, 2010.

R. Letz.
Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas.
In U. Egly and C. G. Fermüller, editors, *TABLEAUX*, volume 2381 of *LNCS*, pages 160–175. Springer, 2002.

M. Samer and S. Szeider.
Backdoor Sets of Quantified Boolean Formulas.
Journal of Automated Reasoning (JAR), 42(1):77–97, 2009.

L. Zhang and S. Malik.
Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation.
In P. Van Hentenryck, editor, *CP*, volume 2470 of *LNCS*, pages 200–215. Springer, 2002.