Dependency Schemes and Search-Based QBF Solving: Theory and Practice

Florian Lonsing

Friday, 27th April, 2012

Quantified Boolean Formulae (QBF):

- Extension of propositional logic.
- PSPACE-completeness (propositional logic: NP-completeness).
- Applications in verification and MC: compact encodings.

This Work:

- QBF solving: variable dependencies.
- Dependency schemes to improve QBF solvers.
- DepQBF: search-based QBF solver, integrates dependency schemes.

QBFEVAL'10 (568 formulae) – without preprocessing		
	Solved	*Avg. Time*
DepQBF	**372**	**334.60**
QuBE7.2-nopp	319	431.47
Nenofex	211	573.65
Quantor 3.0	203	590.15
squolem 2.02	124	708.80

QBFEVAL'10 score-based ranking	
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

Propositional Logic (SAT):

- Boolean variables $V := \{x_1, \ldots, x_n\}$, literals $l := v$ and $l := \overline{v}$ for $v \in V$.
- Clauses $C_i := (l_1 \vee \ldots \vee l_{k_i})$, CNF $\phi := \bigwedge_{i:=1}^{m} C_i$.

Quantified Boolean Formulae (QBF):

- Prenex CNF: quantifier-free CNF over quantified Boolean variables.
- PCNF $F := Q_1 x_1 \ldots Q_n x_n.\ \phi$, where $Q_i \in \{\exists, \forall\}$, no free variables.
- $Q_i x_i \leq Q_{i+1} x_{i+1}$: variables are linearly ordered.
- Applications: compact encodings, e.g. bounded model checking (BMC).

QBF Semantics: recursively based on formula structure.

- $\forall x \phi$ is satisfiable iff both $\phi[x/0]$ and $\phi[x/1]$ are satisfiable.
- $\exists x \phi$ is satisfiable iff $\phi[x/0]$ or $\phi[x/1]$ is satisfiable.
- Related to search-based QDPLL algorithm (see later).

Problem: prefix ordering might limit the freedom in QBF solving.

Variable Dependencies

Semantical Evaluation:

- $Q_1 x_1 \ldots Q_n x_n.\ \phi$: must assign variables in prefix ordering *in general*.
- $\exists a \forall x, y \exists b.\ \phi$: assigning b is possible as soon as x and y are assigned.

Example (Depending Variables)

- $\forall x \exists y.\ (x = y)$ 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.

Breaking the prefix ordering might yield unsound results!

Example (Independent Variables)

- $\forall x \exists y.\ (x \vee \neg y) \wedge (\neg x \vee \neg y)$ is satisfiable: assign x, then y.
- $\exists y \forall x.\ (x \vee \neg y) \wedge (\neg x \vee \neg y)$ is satisfiable: assign y, then x.

Breaking the prefix ordering might be sound and increase freedom!

Dependency Representation

Goal: identify *independent* variables in a given PCNF.

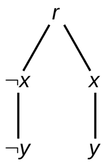
- x and y are independent if they can be assigned in arbitrary order.
- Can we go from linear prefix ordering to *partial ordering* on variables?

Dependency Schemes: relation $D \subseteq (V \times V)$. [SS09, Bie04, BB07, Ben05]

- General framework for expressing (in)dependence in a given PCNF.
- $(x, y) \notin D$: y independent from x.
- $(x, y) \in D$: *conservatively* regard y as depending on x.
- Interpret D as a partial ordering on the variables in general.
- Interesting cases: $(x, y) \notin D$ and $(y, x) \notin D$.

Assignment Trees:

- Theoretical foundation of dependency schemes.
- Tree-like models of PCNFs.
- Represent choice of values for $\exists$-variables.
- Explain variable independence.



$\forall x \exists y. (x \vee \neg y) \wedge (\neg x \vee y).$

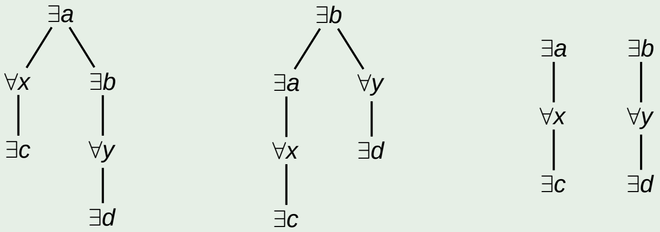
Constructing Dependency Schemes

Syntactic Approaches: tradeoff quality vs. efficiency of computation.

- Trivial dependency scheme D^{triv}: given quantifier prefix.
- Quantifier trees D^{tree}: non-deterministic mini-scoping.
- Standard dependency scheme D^{std}: connections between variables.
- $D^{\text{std}} \subseteq D^{\text{tree}} \subseteq D^{\text{triv}}$: apply D^{std} in practice.

Example (D^{tree} *vs.* D^{std})

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



Either $(a, y) \in D^{\text{tree}}$ or $(b, x) \in D^{\text{tree}}$ but $(a, y) \notin D^{\text{std}}$ and $(b, x) \notin D^{\text{std}}$.

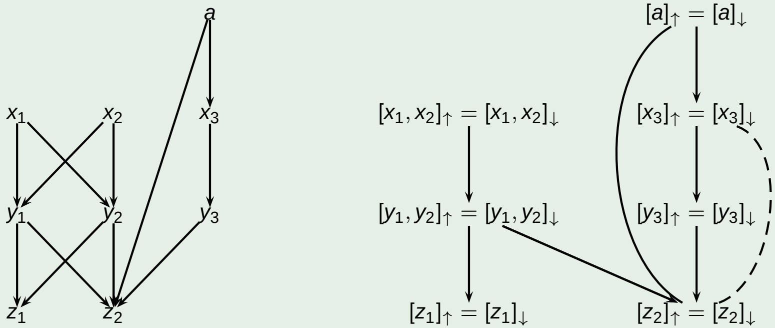
Dependency Scheme D as Directed-Acyclic Graph (DAG):

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

Compressed Dependency Graphs: equivalence relations, aux. edges.

- "Outgoing" edges: $x \approx_{\downarrow} y$ iff $D(x) = D(y)$.
- "Incoming" edges: $x \approx_{\uparrow} y$ iff $D^{-1}(x) = D^{-1}(y)$.
- Efficient algorithm to compute graph for D^{std} (see later).

Example



```
State qdpll ()
  while (true)
    State s = qbcp ();
    if (s == UNDET)
      // 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_constraint (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);
    R' = get_antecedent (p);
    R = constraint_res (R, p, R');
  add_to_formula (R);
  return get_asserting_level (R);
```

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

Backtracking Search with Constraint Learning:

- Classical QDPLL based on quantifier prefix, i.e. D^{triv}.
- `qbcp`: propagate implied (i.e. necessary) assignments.
- `select_dec_var`: decision making.
- `analyze_leaf`: add learned constraint produced by Q-resolution.

QDPLL with Dependency Schemes

```
State qdpll ()
  while (true)
    State s = qbcp ();
    if (s == UNDET)
      // 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_constraint (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);
    R' = get_antecedent (p);
    R = constraint_res (R, p, R');
  add_to_formula (R);
  return get_asserting_level (R);
```

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

Replacing D^{triv} with Arbitrary Dependency Scheme $D \subseteq D^{\mathrm{triv}}$:

- Same basic framework: considering D as a parameter of QDPLL.
- Only change: D used for dependency checking and decision making.
- Expecting more implications, shorter learned constraints.
- Expecting more freedom for selecting decision variables.

Example: Dynamic Effects of Dependency Schemes in QDPLL

Constraint Reduction (CR):

Definition

Let D be a dependency scheme. Given a clause C, *constraint reduction* on C by D produces the clause

$$CR_D(C) := C \setminus \{l \in L_\forall(C) \mid \forall l' \in L_\exists(C) : (v(l), v(l')) \notin D\}.$$

- Part of QBCP and Q-resolution for constraint learning.
- Deleting "largest" universal literals: shortens clauses.
- If $D \subset D'$, then CR by D might produce shorter clauses than CR by D'.
- Potentially more unit/empty clauses.

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 $(a, y) \in D^{\text{triv}}$.

Given $D \subseteq D^{\text{triv}}$ where $(a, y) \notin D$: a is reducible in $(x \vee a \vee y)$, yielding $(x \vee y)$.

Dependency Graph for D^{std}: efficient incremental construction.

- Statistics for QBFEVAL'08 set (3328 formulae).
- Max. time 8.11s, avg. time 0.09s.
- Compare: explicit computation times out (900s) on 135 formulae.
- For $x \in V_\forall, x \in V_\exists$, avg. $|D^{\text{std}}(x)| = 19807$ and $|D^{\text{std}}(x)| = 4$.
- Graph compactly represents sets of depending variables.
- Dep. classes/dep. variables: 0.01 and 0.02 for $x \in V_\forall$, $x \in V_\exists$.
- Graph is tightly integrated in QDPLL.

```
State qdpll ()
  while (true)
    State s = qbcp ();
    if (s == UNDET)
      // 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_constraint (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);
    R' = get_antecedent (p);
    R = constraint_res (R, p, R');
  add_to_formula (R);
  return get_asserting_level (R);
```

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

Dependency Schemes in QDPLL: implemented in our solver DepQBF.

- Pays off despite overhead.
- Expect performance increase from more powerful dependency schemes.

Table: Comparing different dependency schemes in QDPLL.

QBFEVAL'08 (3326 formulae)					
	D^{triv}	D^{tree}	D^{std}	QuBE6.6-nopp	QuBE6.6
Solved	1223	1221	**1252**	1106	**2277**
Avg. Time	579.94	580.64	**572.31**	608.97	**302.49**

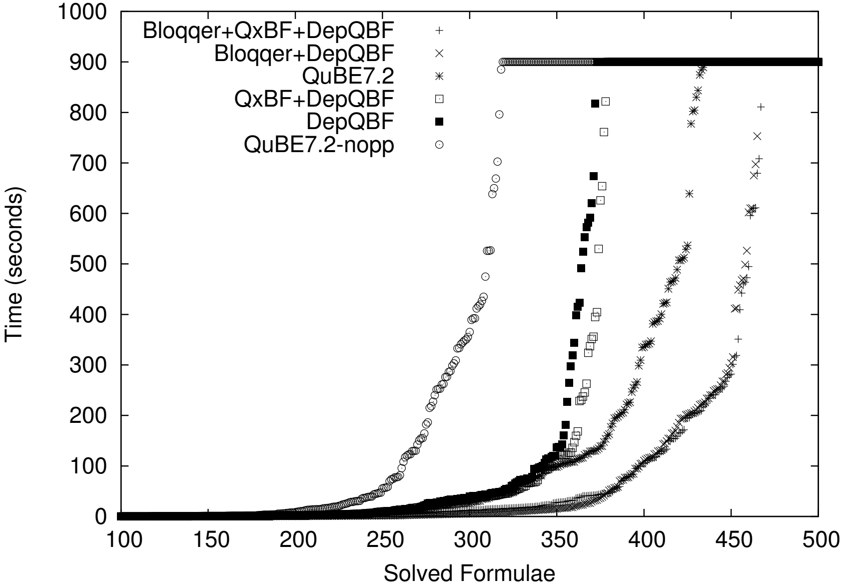
Table: Dynamic effects of different dependency schemes in QDPLL.

QBFEVAL'08 (solved only)						
	$D^{\text{triv}} \cap D^{\text{tree}}$		$D^{\text{triv}} \cap D^{\text{std}}$		$D^{\text{tree}} \cap D^{\text{std}}$	
solved	1172		1196		1206	
time	**23.15**	26.68	**23.73**	25.93	25.63	**22.37**
implied/assigned	90.4%	**90.7%**	88.6%	**90.5%**	90.9%	**92.1%**
backtracks	32431	**27938**	34323	**31085**	**25106**	26136
learnt constr. size	157	**99**	150	**96**	102	**95**

Table: DepQBF and other solvers with and without preprocessing.

	Solved	Avg. Time	SAT	UNSAT
QBFEVAL'10 (568 formulae) – with preprocessing				
Bloqqer + QxBF + DepQBF	468	197.31 (16.47)	224	244
Bloqqer + DepQBF	466	198.50 (7.69)	223	243
QuBE7.2	435	264.70 (–)	202	233
QxBF+ DepQBF	378	323.19 (7.21)	167	211
QBFEVAL'10 (568 formulae) – without preprocessing				
DepQBF	372	334.60	166	206
QuBE7.2-nopp	319	431.47	144	175
Nenofex	211	573.65	103	108
Quantor 3.0	203	590.15	99	104
squolem 2.02	124	708.80	53	71

Experiments (5/5)



Drawbacks of Prenex CNF:

- Quantifier prefix limits freedom of QBF decision procedures.
- Linear ordering of variables might often be relaxed.

Dependency Schemes:

- Variable independence: quality vs. efficiency of computation.
- Related to QBF semantics: inherent property.
- From linear to partial orders on variables: increased freedom.
- Relevant for *arbitrary* QBF solvers.

DepQBF: search-based, competitive, open-source.

- Combining QDPLL with D^{std}.
- Improved overall performance despite overhead.
- Fewer backtracks, shorter learnt constraints, more implications.

Open Problems and Future Work:

- Theoretical results related to QDPLL with $D \subseteq D^{\text{triv}}$.
- Applying more powerful dependency schemes than D^{std}.
- Constraint learning in QDPLL.

References

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, 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.

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.