Quantified Boolean Formulas: Solving and Proofs

Solving

Dr. Joshua Blinkhorn

Friedrich-Schiller-Universität Jena
https://github.com/JoshuaBlinkhorn/QBF

1

Overview

Solving technologies

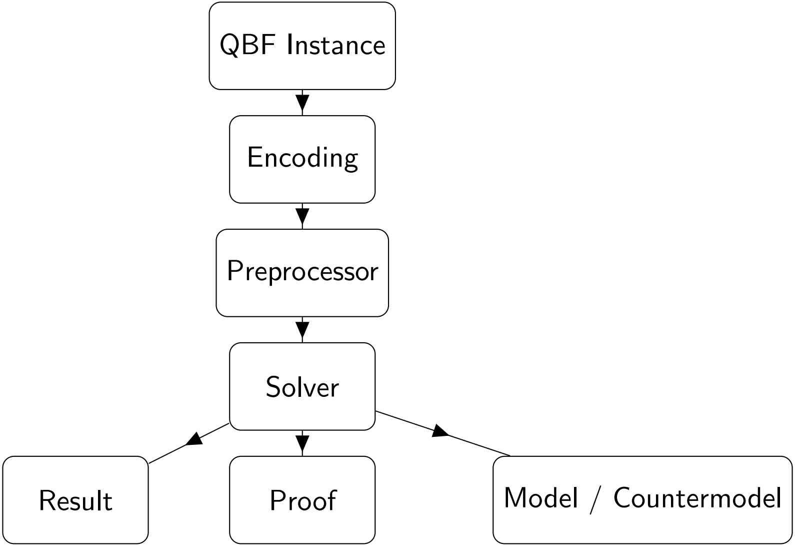
SAT	NP	established efficient technology
QBF	PSPACE	happening now
DQBF	NEXP	in its infancy

Leading Solvers

- In SAT, QCDCL is the dominant solving paradigm
- In QBF, there are various competitive paradigms

Solver	**Paradigm**	**Proof System**
RAReQS	CEGAR	$\forall$Exp+Res (with NP oracle)
CAQE	Clausal Abstraction	Level-ordered Q-Res
Dep-QBF	QCDCL	LD-Q-Res
Dep-QBF	Dependency awareness	Q($\mathcal{D}$)-Res
Qute	Dependency learning	LD-Q-Res

QBF Solving Workflow



The DIMACS CNF Encoding

- machine readable encoding
- variables are natural numbers: $x_1 \mapsto 1$, $x_2 \mapsto 2$ etc.
- negation represented by minus: $\overline{x_1} \mapsto -1$, $\overline{x_2} \mapsto -2$ etc.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_3}) \wedge (\overline{x_1} \vee x_3)$$

```
p cnf 3 4
1 2 3 0
-1 -2 0
-3 0
-1 3 0
```

The QDIMACS Prenex QCNF Encoding

- extends DIMACS
- existential quantifier represented by 'e'
- universal quantifier represented by 'a'

$\exists x_1 \exists x_2 \forall x_3 \exists x_4 \cdot (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_2}) \wedge (\overline{x_3} \vee x_4)$

```
p cnf 4 4
e 1 2 0
a 3 0
e 4 0
1 2 3 0
-1 -3 0
-2 0
-3 4 0
```

2

Preprocessing

Why Preprocess?

- Preprocessors attempt to simplify a QBF while preserving its truth value
- Notion: easier to solve after preprocessing
- Usually, this means reducing the number of variables and the number of clauses
- There are a wide variety of preprocessing techniques
- The proof system QRAT was introduced to cover all of them
- Leading QBF preprocessors: bloqqer and HQS-Pre

Purely Propositional Techniques

- Propositional preprocessing techniques that are logically correct still work for QBFs

- Subsumption:
  $$\mathcal{Q} \cdot (\textstyle\bigwedge_i C_i) \wedge D \wedge E \quad \Rightarrow \quad \mathcal{Q} \cdot (\textstyle\bigwedge_i C_i) \wedge D$$
  provided D is a subclause of E

- Strengthening:
  $$\mathcal{Q} \cdot (\textstyle\bigwedge_i C_i) \wedge (D \vee a) \wedge (E \vee \overline{a}) \quad \Rightarrow \quad \mathcal{Q} \cdot (\textstyle\bigwedge_i C_i) \wedge (D \vee \overline{a}) \wedge E$$
  provided D is a subclause of E

Pure Literal Elimination

- Pure literal elimination is not propositionally logically correct; it only preserves satisfiability
- Works differently for existentials and universals
- Existential version:

$$\mathcal{Q} \cdot (\bigwedge_i C_i) \wedge \bigwedge_j (D_j \vee a) \quad \Rightarrow \quad \mathcal{Q} \cdot (\bigwedge_i C_i)$$

  provided a is existential, $\overline{a}$ doesn't appear in $(\bigwedge_i C_i) \wedge (\bigwedge_j D_j)$
- Universal version

$$\mathcal{Q} \cdot (\bigwedge_i C_i) \wedge \bigwedge_j (D_j \vee a) \quad \Rightarrow \quad \mathcal{Q} \cdot (\bigwedge_i C_i) \wedge \bigwedge_j (D_j)$$

  provided a is universal, $\overline{a}$ doesn't appear in $(\bigwedge_i C_i) \wedge (\bigwedge_j D_j)$

Unit Literal Elimination

- Unit literal elimination is also not propositionally logically correct; but it does preserve satisfiability
- It can only be applied on existential unit clauses:

$$\mathcal{Q} \cdot (\textstyle\bigwedge_i C_i) \wedge (a) \quad \Rightarrow \quad (\mathcal{Q} \cdot \textstyle\bigwedge_i C_i)[\alpha]$$

  provided a is existential, and α is the smallest assignment satisfying a

- Any QBF containing a universal unit clause is false

Universal Reduction

- Universal reduction is logically correct in terms of QBF models
- So it preserves QBF truth value

  $$\mathcal{Q} \cdot (\bigwedge_i C_i) \wedge (D \vee a) \quad \Rightarrow \quad \mathcal{Q} \cdot \bigwedge_i (C_i) \wedge D$$

  provided a is universal, and $\text{var}(a)$ is quantified after all existentials in D, and $(D \vee a)$ is not a tautology
- As a consequence: we can often assume that the final block of a QBF with a CNF matrix is existentially quantified

Blocked Clause Elimination

- Blocked clauses play a key role in SAT preprocessing
- It is an example of a redundancy property
- A redundancy property defines clauses that can be removed (or added) to a CNF while preserving satisfiability
- Propositionally, clause B is blocked w.r.t. a CNF F if B contains a literal for which all resolvents with F are tautologies
- The quantified version again requires a tweak:

$$\mathcal{Q} \cdot (\textstyle\bigwedge_i C_i) \wedge (D \vee a) \quad \Rightarrow \quad \mathcal{Q} \cdot (\textstyle\bigwedge_i C_i)$$

provided a is existential, and for all C_i containing $\overline{a}$, $C_i \otimes_{\overline{a}} D$ has complimentary literals in a variable left of var(a)

Blocked Literal Elimination

- This is the universal analogue of blocked clause elimination
- It allows a universal literal to be removed from a clause:

$$\mathcal{Q} \cdot (\bigwedge_i C_i) \wedge (D \vee a) \quad \Rightarrow \quad \mathcal{Q} \cdot (\bigwedge_i C_i) \wedge D$$

  provided a is universal, and for all C_i containing $\overline{a}$, $C_i \otimes_{\overline{a}} D$ has complimentary literals in a variable left of $\mathrm{var}(a)$

- In contrast to universal reduction, the removed literal is not necessarily right of all existential in the clause

Covered Literal Addition

- Preoprocessors sometimes add literals to clauses
- This can actually be useful - for example, it may increase the set of models for a true QBF
- Covered literal addition

$$\mathcal{Q} \cdot (\bigwedge_i C_i) \wedge (D \vee a) \quad \Rightarrow \quad \mathcal{Q} \cdot (\bigwedge_i C_i) \wedge (D \vee a \vee b)$$

provided a is existential, var(b) is left of var(a), and for all C_i containing $\overline{a}$, either :

  - b is in C_i, or
  - $C_i \vee D$ has complimentary literals in a variable left of var(a)

Existential Variable Elimination

- A method of removing existential variables in the final block
- Based on DP Resolution (Davis-Putnam)
- Propositionally:
  1. take a CNF F
  2. choose a variable x
  3. add all resolvents over x to F
  4. remove all clauses containing x
- This process preserves satisfiability - so it forms a CNF decision procedure
- For QBF, it can be performed on existentials in the final block while preserving truth value
- Hence, it forms a decision procedure for QBF in combination with universal reduction

Universal Expansion

- Expansion of single universal variables preserves truth value
- Preprocessors may perform some universal expansions where it is considered beneficial
- This is a form of partial expansion (but it is not a partial expansion w.r.t. a subset of total universal assignments)
- Guided by heuristics

Ownership and Acknowledgement

- In many cases, the QBF is solved completely in preprocessing
- This raises the question of acknowledgement - for example, in competitions (QBFEVAL)
- Janota: “I used MiniSAT and the C compiler!”

3

Expansion-based Solving

Recap

- The expansion of a QBF $\Phi = \mathcal{Q} \cdot F$ is the CNF

$$\exp(\Phi) := \bigcup_{\alpha \in \langle \mathsf{vars}_{\forall}(\Phi) \rangle} F\left[\alpha \cup \left\{x \mapsto x_{\alpha \restriction L(x)} : x \in \mathsf{vars}_{\exists}(\Phi)\right\}\right]$$

- The partial expansion of a QBF $\Phi = \mathcal{Q} \cdot F$ w.r.t. a set of universal assignments $R \subseteq \langle \mathsf{vars}_{\forall}(\Phi) \rangle$ is the CNF

$$\exp(\Phi, R) := \bigcup_{\alpha \in R} F\left[\alpha \cup \left\{x \mapsto x_{\alpha \restriction L(x)} : x \in \mathsf{vars}_{\exists}(\Phi)\right\}\right]$$

- The partial expansion may be unsatisfiable even when $R \subset \langle \mathsf{vars}_{\forall}(\Phi) \rangle$ is a proper subset of universal assignments

Basic Expansion Decision Procedure

- Arguably the easiest way to solve a QBF Φ:
  1. Write $\exp(\Phi)$ in DIMACS
  2. Pass it to a SAT solver
- Benefit: easy implementation - all work done by SAT solver
- SAT solver employed as an NP oracle
- Drawback: expansion is expensive
- Just computing the expansion takes exponentional time if there are linearly many universal variables, even if the expansion is small

$$\exp(\forall u_1 \cdots \forall u_n \cdot \top) = \top$$

- It makes sense to work with partial expansions

Benders Decomposition

- A techinque for solving linear programming problems
- Exploits block structure of a problem (variable set can be partitioned)
- Divide-and-conquer approach:
  - Divide variables into two sets A and B
  - Solve the *master problem* over A
  - For each candidate solution to the master problem, solve a *subproblem* over B
  - If the subproblem is insoluble, generate a *cut* and add it to the master problem
  - The cut *rules out* the candidate: it will not be selected again
  - Resolve the master problem until no more cuts can be added

Basic Benders Decomposition Approach to QBF

- Consider a QBF $\Phi := \forall U \exists X \cdot F$
- A *winning move* for Φ is $\alpha \in \langle U \rangle$ such that $F[\alpha]$ is unsatisfiable
- Goal: find a winning move for Φ, if one exists
  1. Maintain a set of moves $A \subseteq \langle U \rangle$, initally empty
  2. Find a move $\alpha \in \langle U \rangle$ not in A
  3. Determine whether $F[\alpha]$ is satisfiable with a SAT solver
  4. If not, return α
  5. If so, add α to A
  6. If $A \neq \langle U \rangle$, goto line 2
- Drawback: if Φ is true, all assignments in $\langle U \rangle$ *will* be tested
- In other words: total universal expansion of Φ is constructed
- No information from subproblem passed to master problem

An Extreme Example

- Consider what would happen with this QBF

  $\forall u \forall v \exists x \cdot (u \vee v \vee x) \wedge (u \vee \overline{v} \vee x) \wedge (\overline{u} \vee v \vee x) \wedge (\overline{u} \vee \overline{v} \vee x)$

- Under *every* assignment to $\{u, v\}$, matrix satisfied by $x \mapsto 1$
- SAT solver outputs this in *each* of the four subproblems

- Satisfying assignment to a subproblem explains why a candidate move fails
- We also call this a *counterexample* for the candidate
- In this case, it happens to be the same counterexample for each candidate
- Idea: add counterexamples back into the master problem

Benders Decomposition Done Better

- Find a winning move for $\Phi := \forall U \exists X \cdot F$

  1. Maintain a set of CNFs A in the variables U, initally empty
  2. Find a candidate move $\alpha \in \langle U \rangle$ that falsifies all CNFs in A
  3. Determine whether $F[\alpha]$ is satisfiable with a SAT solver
  4. If not, return α
  5. If so, collect the satisfying assignment β, add $F[\beta]$ to A
  6. Goto line 2

- β is a counterexample to α
- β is also a counterexample to any α' satisfying $F[\beta]$
- Hence, in line 2, if no such move exists, then Φ is true, because every candidate has a counterexample
- The set A is called an *abstraction*

Extreme Example Revisited

- Consider again the QBF

  $\forall u \forall v \exists x \cdot (u \vee v \vee x) \wedge (u \vee \overline{v} \vee x) \wedge (\overline{u} \vee v \vee x) \wedge (\overline{u} \vee \overline{v} \vee x)$

- Regardless of which candidate in $\langle \{u, v\} \rangle$ is chosen first, the counterexample $x \mapsto 1$ is found, and $A = \{\top\}$
- Since $\top$ has no falsifying assignments, we deduce that the QBF is true

- In this case, we only needed to consider a single candidate
- We avoided constructing the total universal expansion
- Essentially, we constructed a partial expansion, whose counterexamples formed a satisfiable abstraction

Quantifiers Exchanged - the Σ_2 Version

- Consider a QBF $\Phi := \exists X \forall U \cdot F$
- A *winning move* for Φ is $\alpha \in \langle X \rangle$ such that $F[\alpha]$ is a tautology
- Goal: find a winning move for Φ, if one exists
  1. Maintain a set of CNFs A in the variables X, initally empty
  2. Find a candidate move $\alpha \in \langle X \rangle$ that satisfies all CNFs in A
  3. Determine whether $F[\alpha]$ is a tautology with a SAT solver
  4. If so, return α
  5. If not, collect the falsifying assignment β, add $F[\beta]$ to A
  6. Goto line 2
- β is a counterexample to α and any α' falsifying $F[\beta]$
- Hence, in line 2, if no such move exists, then Φ is false, because every candidate has a counterexample

Connections to Countermodels

- For a false Σ_2 QBF $\Phi := \exists X \forall U \cdot F$, the set of counterexamples forms the range of a countermodel
  - Why? every candidate has a counterexample *amongst those encoutered*
  - Hence, for each $\alpha \in \langle X \rangle$ a counterexample $\beta \in \langle U \rangle$ was encountered such that $\alpha \cup \beta$ falsifies F
  - In Σ_2, a countermodel is exactly such a mapping
- Hence we must encounter at least $\sigma(\Phi)$ counterexamples, where $\sigma(\Phi)$ is the minimum range of a countermodel for Φ
- Therefore $\sigma(\Phi)$ is a lower bound on the running time of the algorithm
- The final abstraction is essentially the partial expansion of Φ with respect to the set of counterexamples discovered

CEGAR Solving

- CEGAR: Counterexample-guided Abstraction Refinement
- A form of Benders decomposition for solving QBF
- Block structure from quantifier prefix: $\forall U_1 \exists X_1 \cdots \forall U_n \exists X_n$
- A leading CEGAR solver: RAReQs by Janota

Multi-Games

- Merely convenient notation for the pseudocode
- Definition: A multi-game is an expression of the form $QZ \cdot \{\Phi_1, \ldots, \Phi_n\}$ where
  - Q is a quantifier and Z is a block of variables
  - the Φ_i are prenex QBFs whose only free variables are from Z
  - the Φ_i all have the same prefix $\mathcal{Q}$
  - the first quantifier of $\mathcal{Q}$ (if it is not the empty prefix) is opposite to Q
  - the variables of $\mathcal{Q}$ are disjoint from Z
- A winning move for a multigame is an assignment $\alpha \in \langle Z \rangle$ such that
  - if $Q = \exists$, all $\Phi_i[\alpha]$ are true
  - if $Q = \forall$, all $\Phi_i[\alpha]$ are false
- Without loss of generality: assume final block is existential

RAReQs Pseudocode

Function: RAReQs($QZ \cdot \{\Phi_1, \ldots, \Phi_n\}$)
Output: A winning move for Q, or NULL if none exist

if Φ_i have no quantifiers **then return** $\texttt{SAT}(\bigwedge_i \Phi_i)$
$A \leftarrow \emptyset$
$\Psi \leftarrow QZ \cdot A$ `// form initial empty abstraction`
while `true` **do**
  $\alpha' =$ RAReQs(Ψ) `// seek a winning move for the abstraction`
  if $\alpha' =$ NULL **then return** NULL
  $\alpha \rightarrow \alpha' \restriction_Z$ `// filter a move for Z`
  for $i \in [n]$ do $\mu_i \leftarrow$ RAReQS($\Phi_i[\tau]$) `// look for a counterexample`
  if $\mu_i =$ NULL for all $i \in [n]$ **return** τ
  let $i \in [n]$ such that $\mu_i \neq$ NULL
  Remove QZ from the prefix of Φ_i
  $A \leftarrow A \cup \{\Phi_i[\mu_i]\}$ `// refine the abstraction`
end

The Key to RAReQS' Success

- According to the author, RAReQS is based on $\forall$Exp+Res
- A formal proof that an $\forall$Exp+Res refutation can be extracted from the solver trace on a false QBF has not been given
- RAReQs is arguably most successful expansion-based solver
- Key to success: abstraction limits the amount of expansion
- Building the abstraction and solving it is a serious overhead
- Trade-off against the benefit of partial expansion appears favourable for low levels of polynomial heirarchy

RAReQS and Countermodels

- Consider RAReQS on a false QBF Φ
- Imagine the winning moves found for each universal block, concatenated with those from the recursive calls
- This generates a set S of total universal assignments
- S is the range of a countermodel

- Suggestion: RAReQS based on $\forall$Exp+Res with an NP oracle
- Hence minimal countermodel range $\sigma(\Phi)$ is a lower bound for the algorithm running time
- Corollary: equality formulas should be hard for RAReQs

Drawbacks of the RAReQS Approach

- Performs well on QBFs with few blocks
- Preforms worse on QBFs with many blocks
- Candidate moves are winning moves for the abstraction – which is a QBF solved recursively
- Improvement: make abstraction simpler to solve

4

Clausal Abstraction Solving

Clausal Abstraction Solving

- Consider a QBF $\exists X \mathcal{Q} \cdot F$
- If a move $\alpha \in \langle X \rangle$ is not winning, then $\mathcal{Q} \cdot F[\alpha]$ is false
- Idea: record the set of clauses $F[\alpha]$
- Any move $\beta \in \langle X \rangle$ that satisfies none of the clauses in $F[\alpha]$ is not winning either

Example

$$\Phi := \exists x \exists y \forall u \exists z \cdot (x \vee u \vee z)_1 \wedge (x \vee u \vee \overline{z})_2 \wedge (\overline{x} \vee y)_3 \wedge (\overline{y} \vee z)_4$$

- Consider the assignment $\alpha = x \mapsto 0, y \mapsto 0$
- α satisfies clauses 3 and 4, but not clauses 1 and 2
- Moreover, restriction by α yields a false QBF:

$$\Phi[\alpha] := \forall u \exists z \cdot (u \vee z)_1 \wedge (u \vee \overline{z})_2$$

- Observation: any winning move must satisfy *at least one* of clauses 1 and 2

- Consider the assignment $\beta = x \mapsto 0, y \mapsto 1$
- β satisfies neither clause 1 nor 2
- Hence β is not a winning move

The Initial Abstraction

- In CA, the abstraction is a propositional formula
- We consider the existential case first:

$$\Phi := \exists X\, \mathcal{Q} \cdot C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

- The initial abstraction is the CNF

$$\mathsf{abs}(\Phi) := \textstyle\bigwedge_{i \in [k]} C_i \restriction_X \vee b_i$$

- The b_i are fresh variables that do not appear in Φ

$$\Phi := \exists x \exists y \forall u \exists z \cdot (x \vee u \vee z)_1 \wedge (x \vee u \vee \overline{z})_2 \wedge (\overline{x} \vee y)_3 \wedge (\overline{y} \vee z)_4$$

$$\mathsf{abs}(\Phi) := (x \vee b_1) \wedge (x \vee b_2) \wedge (\overline{x} \vee y \vee b_3) \wedge (\overline{y} \vee z \vee b_4)$$

Abstraction Refinement

$$\Phi := \exists X \mathcal{Q} \cdot C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

$$\mathsf{abs}(\Phi) := \bigwedge_{i \in [k]} C_i\restriction_X \vee b_i$$

- Main idea: if $\alpha \in \langle X \rangle$ does not satisfy C_i, the corresponding literal b_i propagates in the abstraction
- Suppose that α turns out to be a losing move:
  - collect the all the literals b_i that propagated, say $b_{i_1}, \ldots, b_{i_m}$
  - add the clause $(\overline{b}_{i_1} \vee \cdots \vee \overline{b_{i_m}})$ to the abstraction
- This forces future candidates to satisfy at least one of the clauses that α did not

The Universal Case

- If the first block is universal:

$$\Phi := \forall U \mathcal{Q} \cdot C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

- The initial abstraction is the CNF

$$\mathsf{abs}(\Phi) := \bigwedge_{i \in [k]} \bigwedge_{a \in C_i \restriction_U} (\overline{a} \vee b_i)$$

$$\Phi := \forall uv \exists x \exists y \cdot (u \vee v \vee x \vee y)_1 \wedge (u \vee \overline{v} \vee x)_2 \wedge (\overline{u} \vee \overline{x} \vee y)_3$$

$$\mathsf{abs}(\Phi) := (\overline{u} \vee b_1) \wedge (\overline{v} \vee b_1) \wedge (\overline{u} \vee b_2) \wedge (v \vee b_2) \wedge (u \vee b_3)$$

Abstraction Refinement in the Universal Case

$$\Phi := \forall U \mathcal{Q} \cdot C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

$$\mathsf{abs}(\Phi) := \bigwedge_{i \in [k]} \bigwedge_{a \in C_i \upharpoonright_U} (\overline{a} \vee b_i)$$

- Main idea: if $\alpha \in \langle U \rangle$ satisfies C_i, the corresponding literal b_i propagates in the abstraction
- Suppose that α turns out to be a losing move:
  - procedure is the same as the existential case
  - collect the all the literals b_i that propagated, say $b_{i_1}, \ldots, b_{i_m}$
  - add the clause $(\overline{b_{i_1}} \vee \cdots \vee \overline{b_{i_m}})$ to the abstraction
- This prevents future candidates from satisfying all of the clauses that α satisfied

Clausal Abstraction Psuedocode

Function: CA($QZ\mathcal{Q} \cdot F$)
Output: The truth value of $QZ\mathcal{Q} \cdot F$

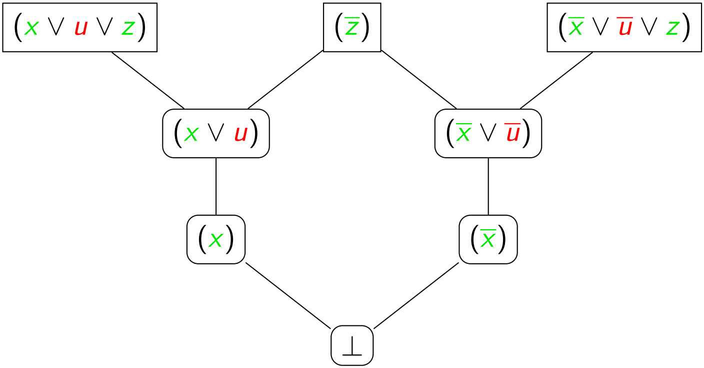
1. **if** $\mathcal{Q}$ is empty **then return** $\texttt{SAT}(F)$
2. $A \leftarrow \mathrm{abs}(\Phi)$ `// adds extention variables` B
3. **while** `true` **do**
4. $\quad$ $(\mathrm{result}, \alpha) = \texttt{SAT}(A)$ `//` $\alpha\restriction_Z$ `is a candidate move`
5. $\quad$ **if** result $=$ UNSAT **then return** $(Q = \exists)$? UNSAT : SAT
6. $\quad$ result $\leftarrow \mathrm{CS}(\mathcal{Q} \cdot F[\alpha\restriction_Z])$ `// recursive call`
7. $\quad$ **if** $Q = \exists$ and result $=$ FALSE **then** $A \leftarrow A \wedge (\overline{\alpha\restriction_B})$ `// refine`
8. $\quad$ **else if** $Q = \forall$ and result $=$ TRUE **then** $A \leftarrow A \wedge (\overline{\alpha\restriction_B})$ `// refine`
9. $\quad$ **else return** $(Q = \exists)$? TRUE : FALSE

Level-ordered Q-Resolution

- Clausal abstraction works recursively, with one abstraction for each quantifier block
- According to its authors, CA corresponds to level-ordered Q-Resolution
- Level-ordered means block-ordered:
  - call a resolution step over an existential pivot x from block X a *step on* X
  - call the reduction of a universal variable u from block U a *step on* U
  - for any block Z: every step on Z precedes all steps on blocks left of Z

Example Level-ordered Q-Resolution Refutation

$$\exists x \forall u \exists z \cdot (x \vee u \vee z) \wedge (\overline{x} \vee \overline{u} \vee z) \wedge (\overline{z})$$



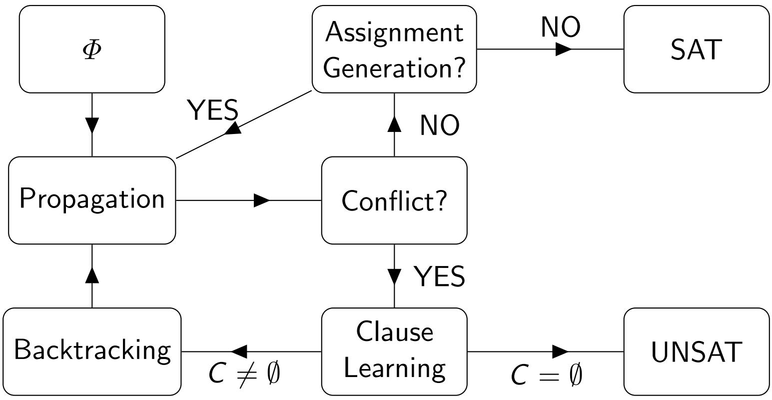
4

QCDCL

Conflict-driven Clause Learning

- State of the art in SAT solving
- Backtracking search of assignment space
- Decisions (variable assignments) and propagations (unit clauses)
- Clause learning - reasons for failed assignments are recorded
- Heuristics (decision, restarts, etc.)

CDCL Workflow



CDCL Psuedocode

Function: CDCL(F)
Output: SAT if F is satisfiable; UNSAT otherwise

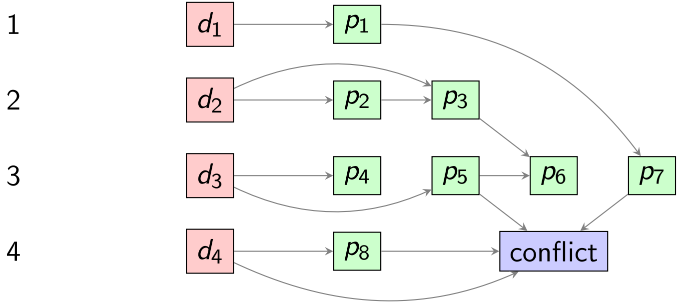
```
while true do
    conflict ← UnitPropagate()
    if conflict == NONE then Decide()
    else
        (clause, level) ← AnalyseConflict(conflict)
        if clause is empty return UNSAT else AddClause(clause)
        Backtrack(level)
```

Conflict Analysis (Clause Learning)

- At conflict, a clause is learned and added to the CNF
- The learned clause is derived by resolution
- If the learned clause is empty, return UNSAT

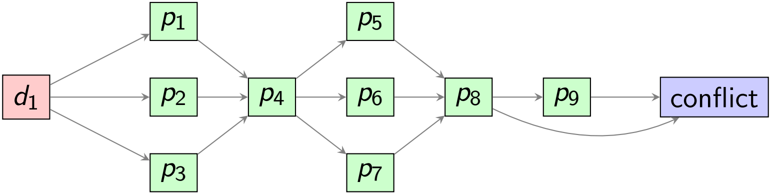
- The resolution process is driven by the implication graph

Clause Learning - Cutting the Implication Graph



Unique Implication Points

- A unique implication point (UIP) is:
  - a node at the highest decision level
  - every path from highest decision to conflict passes through it



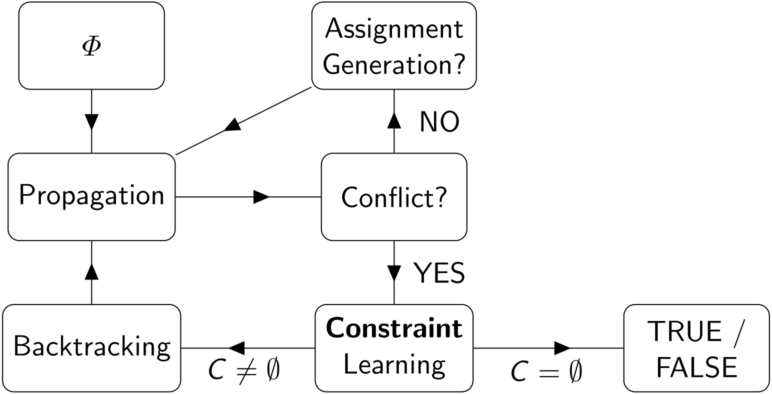
Common Implementation

- advantage: clauses learned from UIPs are always asserting
- asserting means 'becomes unit at a previous decision level'
- the asserting level is second highest decision level in learned clause
- hence: backtrack to asserting level ('backjumping')
- easy implementation:
- resolve conflict clause with 'reason clauses' until there is exactly one variable at highest decision level
- This is the 1UIP learning scheme

What's Different in QCDCL?

1. Cannot terminate when all variables are assigned
   - This means the current assignment satisfies the matrix
   - But to determine truth we need a QBF model
2. Variables cannot be assigned arbitrarily
   - Variable dependence must be resepected
   - A variable can only be assigned after all of its dependencies
   - Scope of decision heuristics limited
3. We have two kinds of constraints
   - Clauses
   - Terms

QCDCL Workflow



What are Terms?

- A term is a conjunction of literals:
$$(a_1 \wedge a_2 \wedge \cdots \wedge a_k)$$
- Terms are in natural correspondence with assignments: there is a unique smallest assignment satisfying a term
$$(x \wedge \overline{y} \wedge \overline{z}) \quad \sim \quad x \mapsto 1, y \mapsto 0, z \mapsto 0$$
- Similarly, there is a unique largest term satisfied by an assignment
- So we can think of terms and assignments as the same
- A term *satisfies* a CNF if the corresponding assignment does

Using Terms to Prove True QBFs

- Q-Resolution is a refutational proof system: it refutes false QBFs
- Dual proof system: Q-Consensus
- Proves true QBFs
- Operates on terms instead of clauses
  - Resolution over universal pivots
  - Existential reduction

Q-Consensus

- Consider a QBF $\mathcal{Q} \cdot F$

axiom:	$\dfrac{}{T}$	T is a term satisfying F
resolution:	$\dfrac{T \wedge u \qquad S \wedge \overline{u}}{T \wedge S}$	T and S are terms u is a universal variable $T \wedge S$ is non-contradictory
weakening:	$\dfrac{T}{T \wedge S}$	T and S are terms $T \wedge S$ is non-contradictory
existential reduction:	$\dfrac{T \wedge a}{T}$	T is a term a is an existential literal $\mathrm{vars}_{\forall}(T) \subseteq L(\mathrm{var}(a))$

Q-Consensus

Definition: A Q-consensus derivation from a QBF $\mathcal{Q} \cdot F$ is a sequence of terms $\Pi = T_1, \ldots, T_k$ derived with the rules above. We call Π a proof of $\mathcal{Q} \cdot F$ when C_k is the empty term.

- The empty term $\emptyset$ is semantically equivalent to $\top$
- Hence $\mathcal{Q} \cdot \emptyset$ is always a true QBF, and therefore has no countermodels.

Soundness and Completeness

- Q-consensus is sound and complete for true QBFs:
  A QBF has a Q-consensus proof if and only if it is true
- Arguments for soundness and completeness dual to Q-Res:
  - Completeness: form model from axioms, resolve and reduce to get the empty term
  - Soundness: every countermodel of the QBF is a countermodel of every derived term
- Hence a QBF has either a Q-Resolution refutation or a Q-Consensus proof (and not both)

QCDCL Intuition

- QCDCL attempts to generate both a Q-Resolution refutation and a Q-consensus proof of the QBF
  - Due to completeness at least one is generated
  - Due to soundness, this determines the QBF truth value
- Thus QCDCL learns both clauses and terms
- By analogy with SAT solving:

$$\text{SAT} \sim \text{TRUE} \qquad \text{UNSAT} \sim \text{FALSE}$$

- When the current assignment is satisfying (SAT), it is added as a term
- This corresponds to a Q-Consensus axiom

QCDCL Pseudocode

Function: QCDCL(Φ)
Output: Truth value of Φ

```
while true do
    conflict ← ConstraintPropagation()          // clauses and terms
    if conflict == NONE then Decide()           // dependencies apply
    else                                        // some constraint is empty
        (constraint, level) ← AnalyseConflict(conflict)
        if constraint is empty
            if constraint is a clause return FALSE else return TRUE
        AddConstraint(constraint)
        Backtrack(level)
```

Conflict Analysis in QCDCL

- Clauses and terms are derived using Q-Resolution and Q-Consensus analogously to SAT solving
- For UIP learning, we need long-distance derivations
  - Universal reduction is performed as propagation
  - As a result, UIP learning can generate universal tautologies
  - But these tautologies are always right of the pivot
- Hence soundness of long-distance Q-Resolution is crucial to solver performance
- Dual situation for terms: long-distance Q-Consensus

5

Dependency Schemes

Motivations

- Desicion heuristics are central to CDCL performance
- But in QBF, allowable decisions are restricted by the prefix

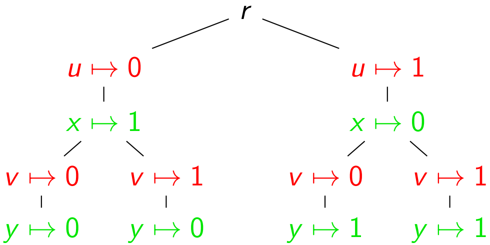
$$\exists x_1 \cdots x_n \forall u_1 \cdots u_n \exists z_1 \cdots z_n \cdot F$$

  - No u_i can be a decision until all x_i have been assigned
  - No z_i can be a decision until all u_i have been assigned
- Reason: variable dependence must be respected
- Consequence: Scope of decision heuristics limited

Spurious Dependencies

- However - dependencies are often spurious
- Spurious dependencies can be safey ignored and the QBF truth value will not change

$$\forall u \exists x \forall v \exists y \cdot \{u, x, v\}, \{u, \bar{y}\}, \{\bar{u}, y\}$$

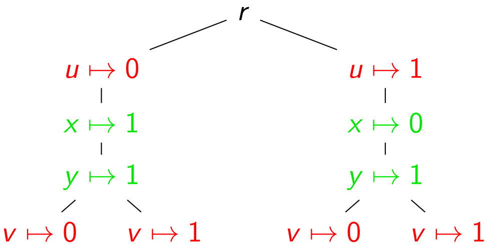


- In this model, y does not depend on v

Spurious Dependencies

- However - dependencies are often spurious
- Spurious dependencies can be safey ignored and the QBF truth value will not change

$$\forall u \exists x \forall v \exists y \cdot \{u, x, v\}, \{u, \bar{y}\}, \{\bar{u}, y\}$$



- In this model, y does not depend on v

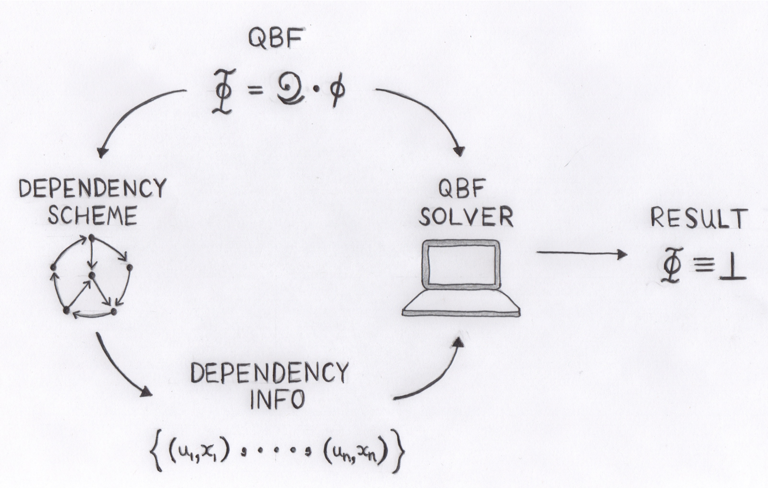
Improving Decision Heuristic via Dependency Analysis

- Identify spurious dependencies before the solving starts
- The dependency sets shrink

$$\forall u \exists x \forall v \exists y \cdot \{u, x, v\}, \{u, \bar{y}\}, \{\bar{u}, y\}$$

  - Here, $L(y) = (u, v)$
  - But we know that y is independent of v
  - So we could take $L(y) = (u)$
  - Then y is available as decision before v is assigned
  - Justification: there exists a model which *exhibits* the independence of y on v
- In general: more variables available for decision
- Scope of decision heuristics is increased

Static Dependency-aware Solving



Variable Dependence in the Equality Formulas

$$EQ_n \quad := \quad \exists x_1 \cdots x_n \forall u_1 \cdots u_n \exists z_1 \cdots z_n \cdot$$
$$\left(\bigwedge_{i \in [n]} (x_i \vee u_i \vee z_i) \right) \wedge \left(\bigwedge_{i \in [n]} (\overline{x_i} \vee \overline{u_i} \vee z_i) \right) \wedge (\bigvee_{i \in [n]} \overline{z_i})$$

- Let us recall the countermodel:

$$\begin{array}{rcccl} h & : & \langle \text{vars}_\exists(EQ_n) \rangle & \rightarrow & \langle \text{vars}_\forall(EQ_n) \rangle \\ & & \alpha & \mapsto & \{u_1 \mapsto \alpha(x_1), \ldots, u_n \mapsto \alpha(x_n)\} \end{array}$$

- Each u_i depends only on x_i
- This dependency structure cannot be written as a QBF prefix; the best we can do is something like this:

$$\exists x_1 \forall u_1 \cdots \exists x_n \forall u_n \exists z_1 \cdots \exists z_n$$

- Still we have spurious dependence of u_n on $x_1, \ldots, x_{n-1}$

A Note About Duality

- In practice, dependency schemes work with two kinds of dependencies:
  - dependence of existentials on universals
  - dependence of universals on existentials
- These are handled in a ‘dual’ fashion
- For simplicity: focus on the first kind only

Dependency Schemes - Traditional Definition

Definition: The *trivial dependency scheme* is the mapping $\mathcal{D}^{\mathsf{trv}}$ that maps a QBF Φ to the set of pairs

$$\mathcal{D}^{\mathsf{trv}}(\Phi) := \{(u, x) : u \in \mathsf{vars}_{\forall}(\Phi), x \in \mathsf{vars}_{\exists}(\Phi), u \in L(x)\}$$

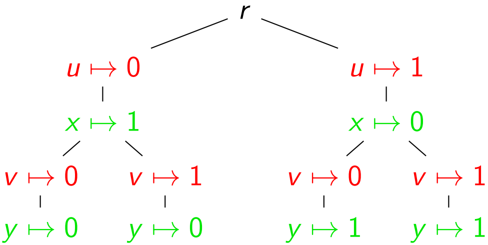
- Total order of prefix represented as set of pairs

Definition: A *dependency scheme* is a mapping that maps each QBF Φ to a subset of $\mathcal{D}^{\mathsf{trv}}(\Phi)$

- Absent pairs should represent spurious dependencies
- Dependency scheme defines a partial order on the variables
- Better approximation to the 'true' dependency structure of Φ

What Do We Want From a Dependency Scheme?

$$\Phi := \forall u \exists x \forall v \exists y \cdot \{u, x, v\}, \{u, \bar{y}\}, \{\bar{u}, y\}$$



- y does not depend on v
- (v, y) is a spurious dependency – we want $(v, y) \notin \mathcal{D}(\Phi)$

Standard Dependency Scheme $\mathcal{D}^{\mathsf{std}}$

- Connection-based dependencies

$$\Phi := \forall u \exists x \exists y \exists z \quad \{u, x\} \qquad \{x, y\} \qquad \{y, z\}$$

$x \qquad y$

(u, z) is a $\mathcal{D}^{\mathsf{std}}$-dependency

- This connection links u to z via connecting variables x and y
- Connecting variables must be:
  - existential
  - right of u (i.e. not in $\mathsf{L}(\mathsf{u})$)
- $(u, z) \in \mathcal{D}^{\mathsf{std}}(\Phi) \sim$ 'z depends on u in Φ according to $\mathcal{D}^{\mathsf{std}}$'
- A dependency is only acknowledged when a connection exists
- Spurious dependencies identified by absence of a connection

Standard Dependency Scheme $\mathcal{D}^{\mathsf{std}}$

$$\Phi := \forall u \exists x \forall v \exists y \cdot \{u, x, v\}, \{u, \bar{y}\}, \{\bar{u}, y\}$$

- There are trivial connnections from u to x and y
- Hence $(u, x), (u, y) \in \mathcal{D}^{\mathsf{std}}(\Phi)$

- There is no connection from v to y
- Hence $(v, y) \notin \mathcal{D}^{\mathsf{std}}(\Phi)$

- $\mathcal{D}^{\mathsf{std}}$ identifies that y does not depend on v in Φ
- That is, (v, y) is a spurious dependency
- A solver using $\mathcal{D}^{\mathsf{std}}$ can assign y before u

Universal Reduction with Dependency Schemes

universal reduction:	$\dfrac{C \vee a}{C}$	C is a clause a is a universal literal $\mathrm{vars}_{\exists}(C) \subseteq L(\mathrm{var}(a))$

- The condition $\mathrm{vars}_{\exists}(C) \subseteq L(\mathrm{var}(a))$ is equivalent to:

$$\text{for all } x \in \mathrm{vars}_{\exists}(C), \;\; (\mathrm{var}(a), x) \notin \mathcal{D}^{\mathrm{trv}}(\Phi)$$

$\dfrac{C \vee a}{C}$	C is a clause a is a universal literal for all $x \in \mathrm{vars}_{\exists}(C)$, $(\mathrm{var}(a), x) \notin \mathcal{D}(\Phi)$

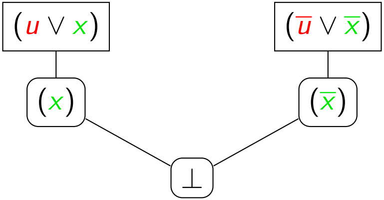
Learning With Dependency Schemes

- Dependency-aware solvers employ universal reduction w.r.t. their dependency scheme $\mathcal{D}$
- Hence, learning mechanism is based on Q-Res with universal reduction w.r.t. $\mathcal{D}$
- We call this proof system Q($\mathcal{D}$)-Res
- Even better: use long-distance resolution – LD-Q($\mathcal{D}$)-Res
- Crucial: soundness
- How can we be sure that Q($\mathcal{D}$)-Res and LD-Q($\mathcal{D}$)-Res are sound?

Unsound Dependency Schemes

- There exist obvious unsound dependency schemes
- E.g. the empty dependency scheme: $\mathcal{D}(\Phi) = \emptyset$ for all QBFs Φ
- Allows all universal reductions

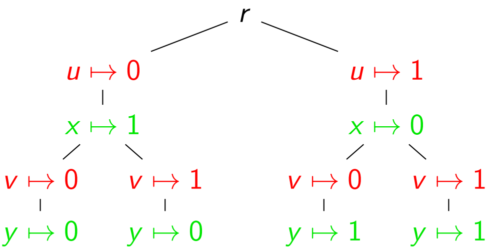
$$\forall u \exists x \cdot (u \vee x) \wedge (\bar{u} \vee \bar{x})$$



- Non-trivial problem: even the so-called 'optimal' scheme $\mathcal{D}^{\text{opt}}$ is unsound!

Defining Independence

$$\Phi := \forall u \exists x \forall v \exists y \cdot \{u, x, v\}, \{u, \bar{y}\}, \{\bar{u}, y\}$$



Definition: An existential x is independent of a universal u w.r.t. a model M of a QBF Φ when flipping the value of u on any path does not change the value of x.

Problems With the Optimal Dependency Scheme

Definition: The optimal dependency scheme $\mathcal{D}^{\mathsf{opt}}$ is defined by

$$\mathcal{D}^{\mathsf{opt}}(\Phi) := \mathcal{D}^{\mathsf{trv}}(\Phi) \setminus O(\Phi)$$

$$O(\Phi) := \{(u, x) : x \text{ independent of } u \text{ in some model of } \Phi\}$$

- Idea: all spurious dependencies are removed
- Problem: unsound
- Crux: different spurious dependencies *may be exhibited by different models*
- Solution: consider a set of spurious depencies exhibited by a single model

Full Exhibition

Definition: Given a dependency scheme $\mathcal{D}$ and a QBF Φ, a $\mathcal{D}$-model for Φ is a model in which x is independent of u whenever $(u, x) \notin \mathcal{D}(\Phi)$.

Definition: A dependency scheme $\mathcal{D}$ is fully exhibited when every true QBF Φ has a $\mathcal{D}$-model.

Theorem: The standard dependency scheme $\mathcal{D}^{\mathsf{std}}$ is fully exhibited.

Full Exhibition is Sufficient for Soundness

Theorem: If $\mathcal{D}$ is fully exhibited, then Q($\mathcal{D}$)-Res and LD-Q($\mathcal{D}$)-Res are sound.

- Proof: follow the soundness proof for Q-Res:
  - Q-Res rules preserve models
  - Q($\mathcal{D}$)-Res rules preserve $\mathcal{D}$-models

Corollary: Q($\mathcal{D}^{\mathsf{std}}$)-Res and LD-Q($\mathcal{D}^{\mathsf{std}}$)-Res are sound.

Theorem: Q($\mathcal{D}^{\mathsf{opt}}$)-Res is not sound.

- Proof breaks down:
  - $\mathcal{D}^{\mathsf{opt}}$-models do not always exist
  - Using different models to validate different reduction steps breaks the induction

Complexity of Dependency Schemes

- $QDRes\mathcal{D}$ always simulates Q-Res $=$ Q($\mathcal{D}^{\text{trv}}$)-Res
- Can dependency schemes shorten QBF proofs (exponentially)?
- Yes..
- However – relative proof complexities of Q-Res and Q($\mathcal{D}^{\text{std}}$)-Res is an open problem (neither simulation nor separation has been proved.)
- Separation shown with a stronger dependency scheme

Reflexive Resolution Path Dependency Scheme $\mathcal{D}^{\text{rrs}}$

- Connection-based dependencies with polarity

$$\{u, x\} \qquad \{\bar{x}, y\} \qquad \{\bar{y}, z\}$$

$$x \qquad\qquad y$$

$$\forall u \exists x \exists y \exists z$$

$$\{\bar{u}, x\} \qquad \{\bar{x}, y\} \qquad \{\bar{y}, \bar{z}\}$$

$$x \qquad\qquad y$$

(u, z) is a $\mathcal{D}^{\text{rrs}}$-dependency

- Resolution paths link u to z *and* $\overline{u}$ to $\overline{z}$ via connecting variables x and y
- Connecting variables must:
  - be existential
  - be right of u (i.e. not in $\mathrm{L}(\mathrm{u})$)
  - appear in opposite polarities
  - be consecutively distinct

Q($\mathcal{D}^{\text{rrs}}$)-Res versus Q-Res (1)

- We prove a stronger result.

  Theorem: Q($\mathcal{D}^{\text{rrs}}$)-Res is exponentially stronger than Q($\mathcal{D}^{\text{std}}$)-Res.

- Use the equality formulas

$$EQ_n \quad := \quad \exists x_1 \cdots x_n \forall u_1 \cdots u_n \exists z_1 \cdots z_n .$$

$$\left(\bigwedge_{i \in [n]} (x_i \vee u_i \vee z_i) \right) \wedge \left(\bigwedge_{i \in [n]} (\overline{x_i} \vee \overline{u_i} \vee z_i) \right) \wedge (\bigvee_{i \in [n]} \overline{z_i})$$

Q($\mathcal{D}^{\text{rrs}}$)-Res versus Q-Res (2)

Lower bound for Q($\mathcal{D}^{\text{std}}$)-Res

- Lemma: $\mathcal{D}^{\text{std}}(\text{EQ}_n)$ is the set of trivial dependencies.
- Q($\mathcal{D}^{\text{std}}$)-Res and Q-Res are equivalent on EQ_n.
- Q-Res refutations are be exponentially large.

Upper bound for Q($\mathcal{D}^{\text{rrs}}$)-Res

- $\mathcal{D}^{\text{rrs}}$ identifies all independencies.
- Lemma: $\mathcal{D}^{\text{rrs}}(\text{EQ}_n) = \emptyset$.
- Allows linear-size refutations of EQ_n.

$\mathsf{Q}(\mathcal{D}^{\mathsf{rrs}})$-Res versus Q-Res (3)

- Short refutations of equality in $\mathsf{Q}(\mathcal{D}^{\mathsf{rrs}})$-Res
  - Since $\mathcal{D}^{\mathsf{rrs}}(\mathsf{EQ}_n) = \emptyset$, all universal literals can be reduced from axioms
  - reduce all universal literals to obtain $(x_i \vee z_i)$ and $(\overline{x_i} \vee z_i)$
  - resolve over all x_i to obtain unit clauses (z_i)
  - resolve unit clauses with $(\overline{z_1} \vee \cdots \vee \overline{z_n})$

Dependency-aware Solving – Some Thoughts

- Scope of decision heuristics and propagation improved
- Trade-off with computation overhead of the scheme
- Overall: faster solving with the standard dependency scheme
- Problems with soundness
- Essence obfuscated?
- Not ideal: dependency scheme *written into proof system rules*

Dependency Quantified Boolean Formulas (DQBF)

- Existential dependencies made explicit

$$\forall u_1 \cdots \forall u_n \exists e_1(U_1) \cdots \exists e_m(U_m) \cdot \phi(u_1, \ldots, u_n, e_1, \ldots, e_m)$$

$$U_i \subseteq \{u_1, \ldots, u_n\}$$

- Semantics: model a set of Boolean functions $f_1, \ldots, f_m$ s.t.
  (a) variables of f_i are U_i
  (b) $\phi(u_1, \ldots, u_n, f_1, \ldots, f_m)$ is a tautology

DQBF proof systems

- Expansion: DQBF version of $\forall$Exp+Res is sound and complete
- Reduction: not so simple
  - Q-Res is sound but incomplete
  - LD-Q-Res is not sound
- Nevertheless - we can use DQBF proof systems to refute QBF
- Even LD-Q-Res
- Result: shorter QBF proofs

DQBF-centric Interpretation of Dependency Schemes

Definition: A dependency scheme is a truth-preserving polynomial-time computable mapping from QBF into DQBF s.t.:

(a) Φ and $\mathcal{D}(\Phi)$ have the same matrix

(b) dependency sets of $\mathcal{D}(\Phi)$ are subsets of those of Φ

- Dependency scheme can be viewed as a preprocessor
- Independent of the solver and the proof system
- Proof complexity of dependency schemes can be studied via fragments of DQBF proof systems
- For example: Q($\mathcal{D}$)-Res is the fragment of Q-Res on the image of $\mathcal{D}$
- In general: P($\mathcal{D}$) is the fragment of P on the image of $\mathcal{D}$
- Soundness for free via truth preservation (full exhibition)
- Completeness should follow from QBF system (*monotonicity*)