Quantified Boolean Formula: Evaluation, Certification, and Applications

Jie-Hong Roland Jiang
National Taiwan University

TAROT Summer School
Saint Petersburg, Russia, June 2011

Outline

- Satisfiability (SAT)
  - Conjunctive Normal Form (CNF)
  - SAT solving and Craig interpolation
  - Application
    - Functional dependency

- Quantified Satisfiability (QSAT)
  - Quantified Boolean Formula (QBF)
  - QBF evaluation and certification
  - Application
    - Relation determinization, program synthesis

Satisfiability

Normal Forms

- A **literal** is a variable or its negation
- A **clause** (**cube**) is a disjunction (conjunction) of literals
- A **conjunctive normal form** (CNF) is a conjunction of clauses; a **disjunctive normal form** (DNF) is a disjunction of cubes
  - E.g.,
    CNF: $(a+\neg b+c)(a+\neg c)(b+d)(\neg a)$
    - $(\neg a)$ is a unit clause, d is a pure literal

    DNF: $a\neg bc + a\neg c + bd + \neg a$

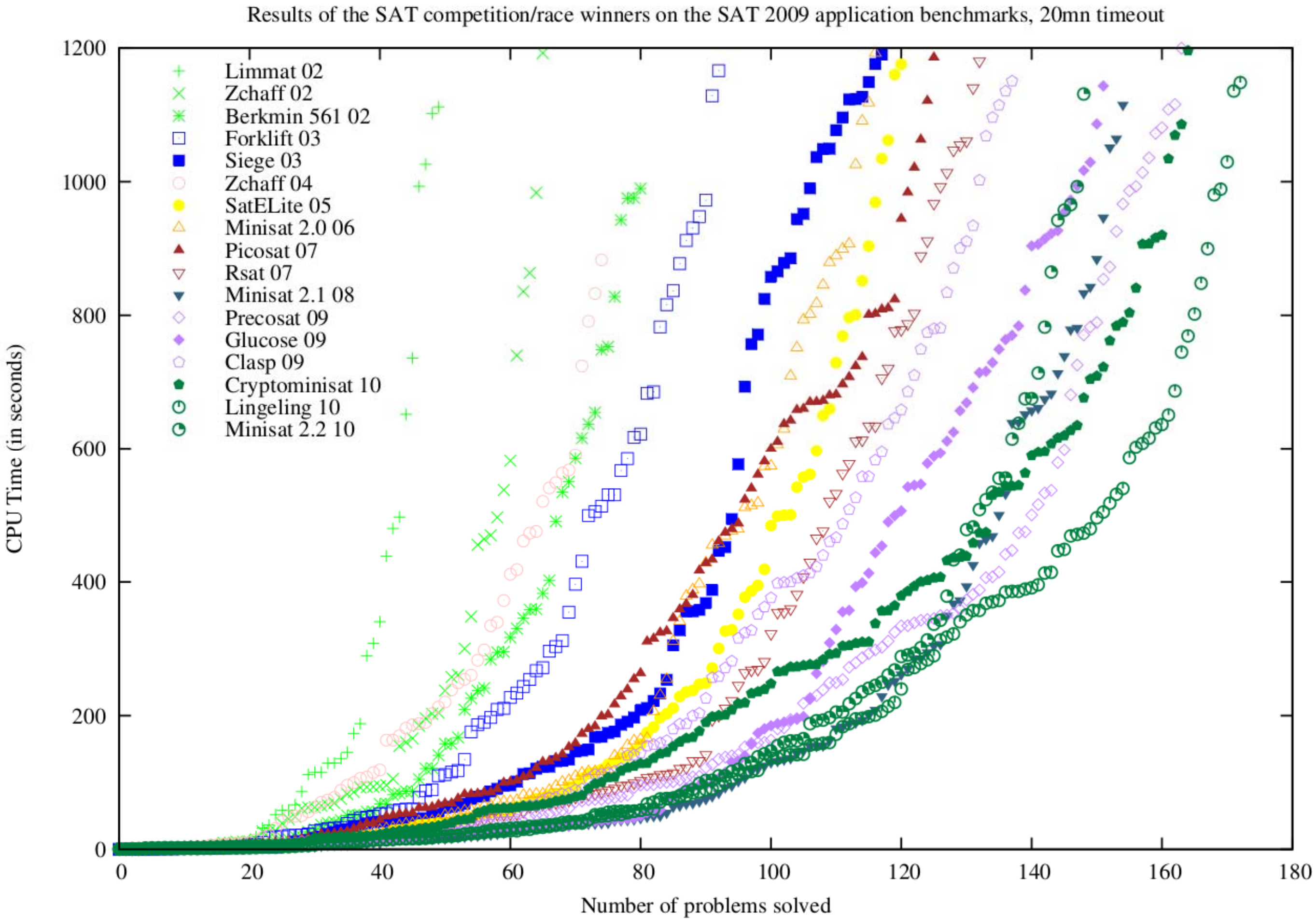
Satisfiability

- The **satisfiability** (SAT) problem asks whether a given CNF formula can be true under some assignment to the variables

- In theory, SAT is intractable
  - The first shown NP-complete problem [Cook, 1971]

- In practice, modern SAT solvers work 'mysteriously' well on application CNFs with ~100,000 variables and ~1,000,000 clauses
  - It enables various applications, and inspires QBF and SMT (Satisfiability Modulo Theories) solver development

SAT Competition



http://www.satcompetition.org/PoS11/

SAT Solving

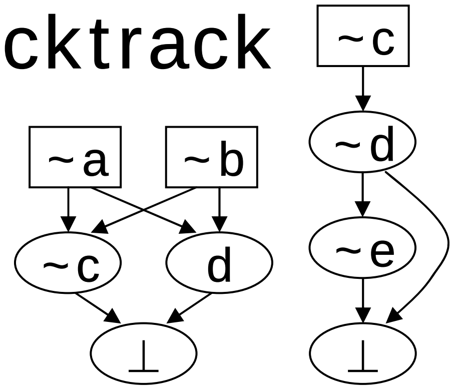
- Ingredients of modern SAT solvers:
  - DPLL-style search
    - [Davis, Putnam, Logemann, Loveland, 1962]
  - Conflict-driven clause learning (CDCL)
    - [Marques-Silva, Sakallah, 1996 (GRASP)]
  - Boolean constraint propagation (BCP) with two-literal watch
    - [Moskewicz, Modigan, Zhao, Zhang, Malik, 2001 (Chaff)]
  - Decision heuristics using variable activity
    - [Moskewicz, Modigan, Zhao, Zhang, Malik, 2001 (Chaff)]
  - Restart
  - Preprocessing
  - Support for incremental solving
    - [Een, Sorensson, 2003 (MiniSat)]

Pre-Modern SAT Procedure

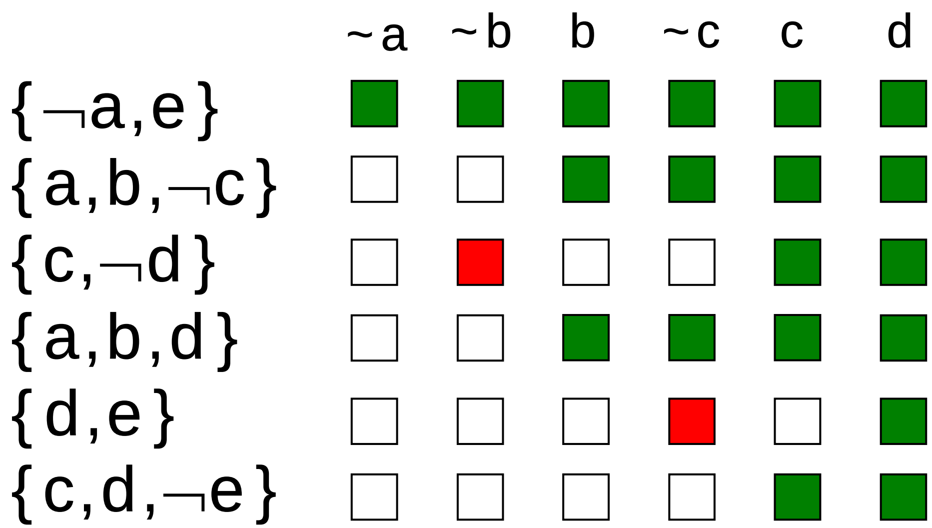
```
Algorithm DPLL(Φ)
{
  while there is a unit clause {l} in Φ
     Φ = BCP(Φ, l);
  while there is a pure literal l in Φ
     Φ = assign(Φ, l);
  if all clauses of Φ satisfied   return true;
  if Φ has a conflicting clause   return false;
  l := choose_literal(Φ);
  return DPLL(assign(Φ,¬l)) ∨ DPLL(assign(Φ,l));
}
```

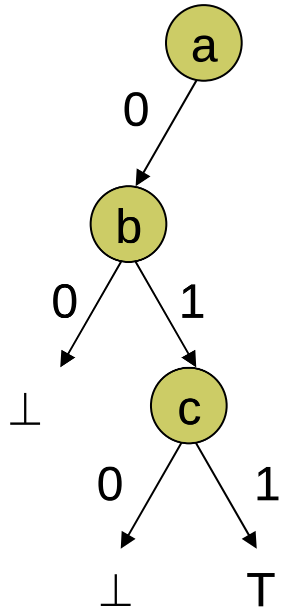
DPLL Procedure

- Chorological backtrack



- E.g.





Modern SAT Procedure

```
Algorithm CDCL(Φ)
{
  while(1)
     while there is a unit clause {l} in Φ
         Φ = BCP(Φ, l);
     while there is a pure literal l in Φ
         Φ = assign(Φ, l);
     if Φ contains no conflicting clause
        if all clauses of Φ are satisfied   return true;
        l := choose_literal(Φ);
        assign(Φ,l);
     else
        if conflict at top decision level   return false;
        analyze_conflict();
        undo assignments;
        Φ := add_conflict_clause(Φ);
}
```

Conflict Analysis & Clause Learning

- There can be many learnt clauses from a conflict
- Clause learning admits non-chorological backtrack

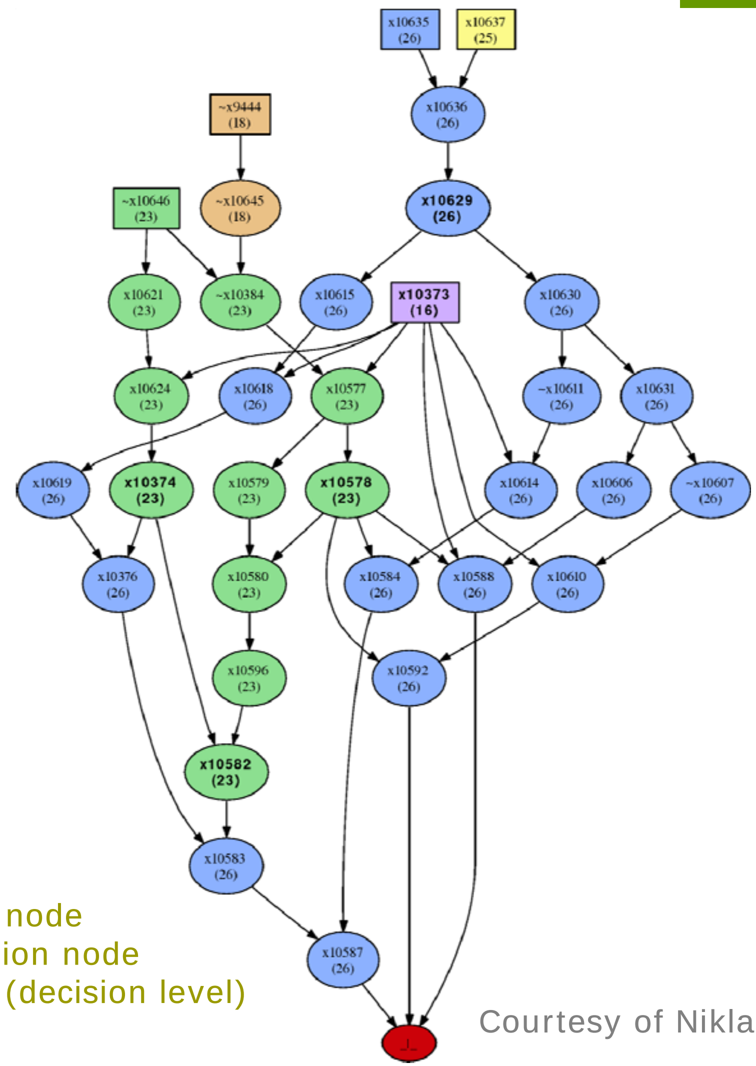
- E.g.,
  {¬x10587, ¬x10588, ¬x10592}

  …

  {¬x10374, ¬x10582, ¬x10578, ¬x10373, ¬x10629}

  …

  {x10646, x9444, ¬x10373, ¬x10635, ¬x10637}



Box: decision node
Oval: implication node
Inside: literal (decision level)

Courtesy of Niklas Een

Clause Learning as Resolution

- **Resolution** of two clauses $C_1 \vee x$ and $C_2 \vee \neg x$:

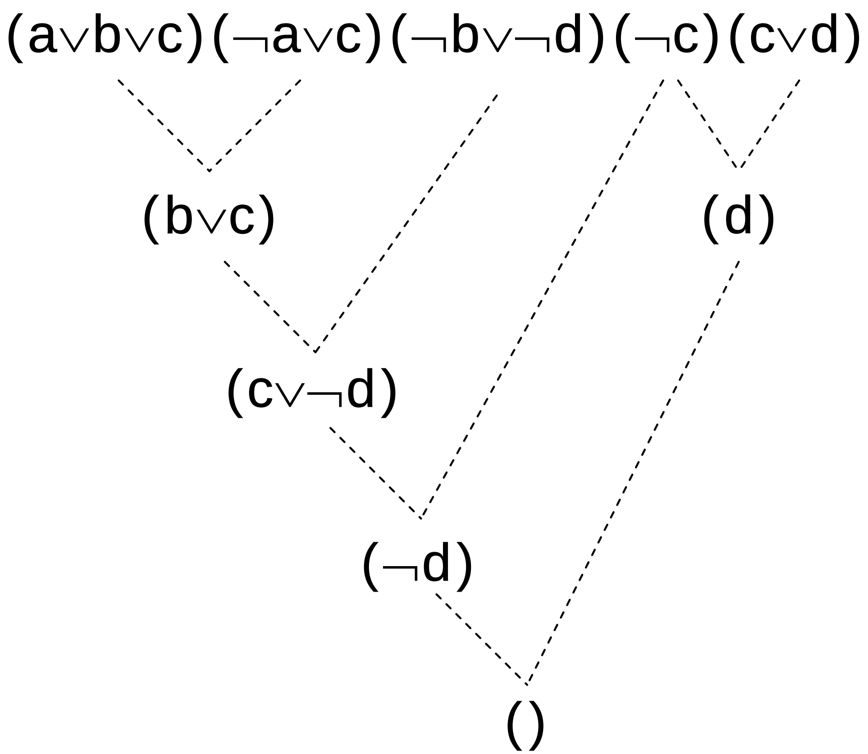
$$\frac{C_1 \vee x \qquad C_2 \vee \neg x}{C_1 \vee C_2}$$

  where x is the **pivot variable** and $C_1 \vee C_2$ is the **resolvant**, i.e., $C_1 \vee C_2 = \exists x.(C_1 \vee x)(C_2 \vee \neg x)$

- A learnt clause can be obtained from a sequence of resolution steps
  - Exercise:
    Find a resolution sequence leading to the learnt clause
    $\{\neg x10374, \neg x10582, \neg x10578, \neg x10373, \neg x10629\}$
    in the previous slides

Resolution

- Resolution is complete for SAT solving
  - A CNF formula is unsatisfiable if and only if there exists a resolution sequence leading to the empty clause
  - Example



SAT Certification

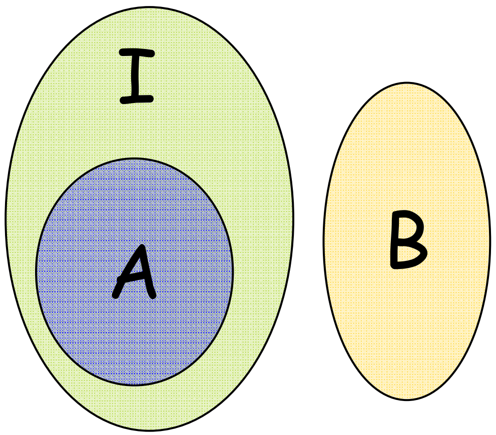
- True CNF
  - Satisfying assignment (model)
    - Verifiable in linear time

- False CNF
  - Resolution refutation
    - Potentially of exponential size

Craig Interpolation

- [Craig Interpolation Thm, 1957]
  If $A \wedge B$ is UNSAT for formulae A and B, there exists an interpolant I of A such that

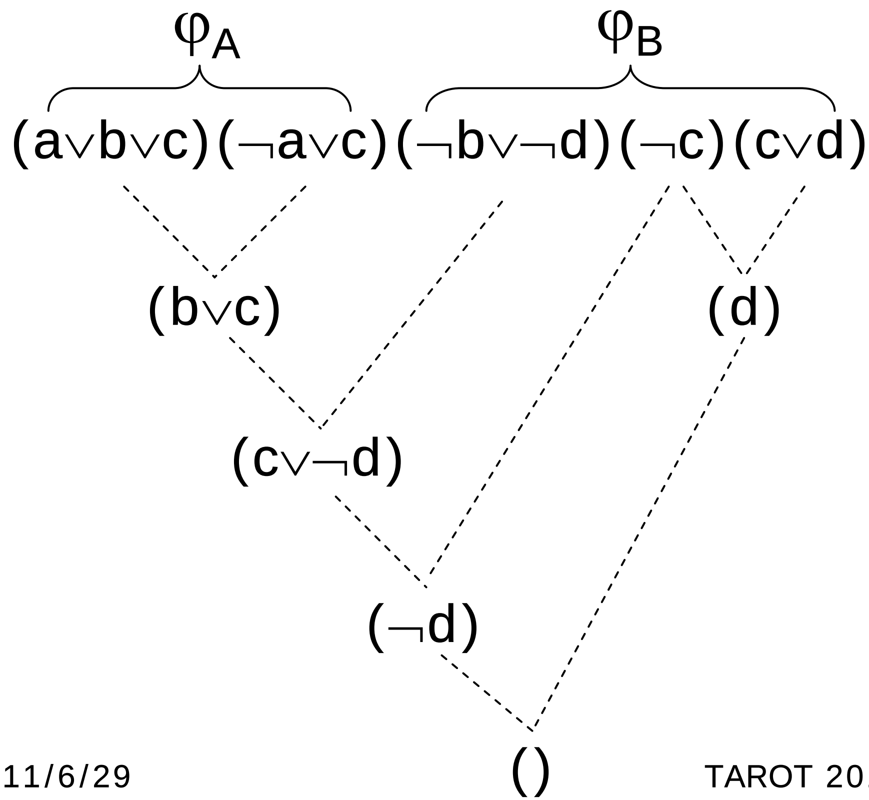
  1. $A \Rightarrow I$
  2. $I \wedge B$ is UNSAT
  3. I refers only to the common variables of A and B

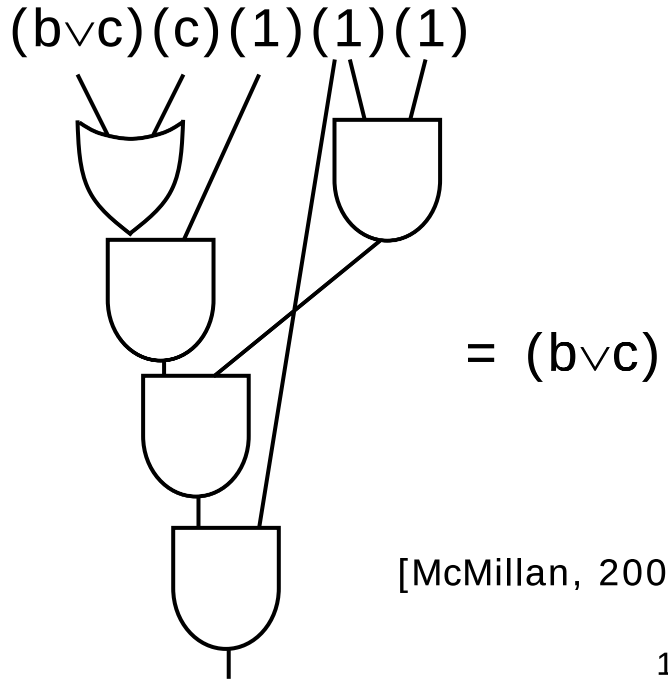


I is an abstraction of A

Interpolant and Resolution Proof

- SAT solver may produce the resolution proof of an UNSAT CNF φ
- For $\varphi = \varphi_A \wedge \varphi_B$ specified, the corresponding interpolant can be obtained in time linear in the resolution proof





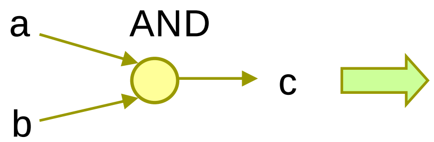
[McMillan, 2003]

Circuit to CNF Conversion

- Circuit to CNF conversion can be done in time linear w.r.t. circuit size [Tseitin, 1968]
  - Trick: introduce intermediate variables
    - The resultant formula can blow up if no intermediate variables are allowed to exist

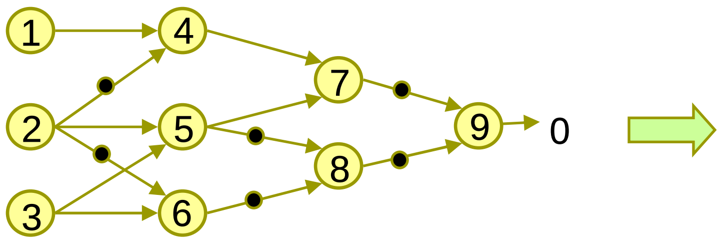
Circuit to CNF Conversion

- Example
  - Single gate:



$(\neg a + \neg b + c)(a + \neg c)(b + \neg c)$

  - Circuit of connected gates:



Is output always 0 ?

Justify to “1”

$(\neg 1 + 2 + 4)(1 + \neg 4)(\neg 2 + \neg 4)$
$(\neg 2 + \neg 3 + 5)(2 + \neg 5)(3 + \neg 5)$
$(2 + \neg 3 + 6)(\neg 2 + \neg 6)(3 + \neg 6)$
$(\neg 4 + \neg 5 + 7)(4 + \neg 7)(5 + \neg 7)$
$(5 + 6 + 8)(\neg 5 + \neg 8)(\neg 6 + \neg 8)$
$(7 + 8 + 9)(\neg 7 + \neg 9)(\neg 8 + \neg 9)$
(9)

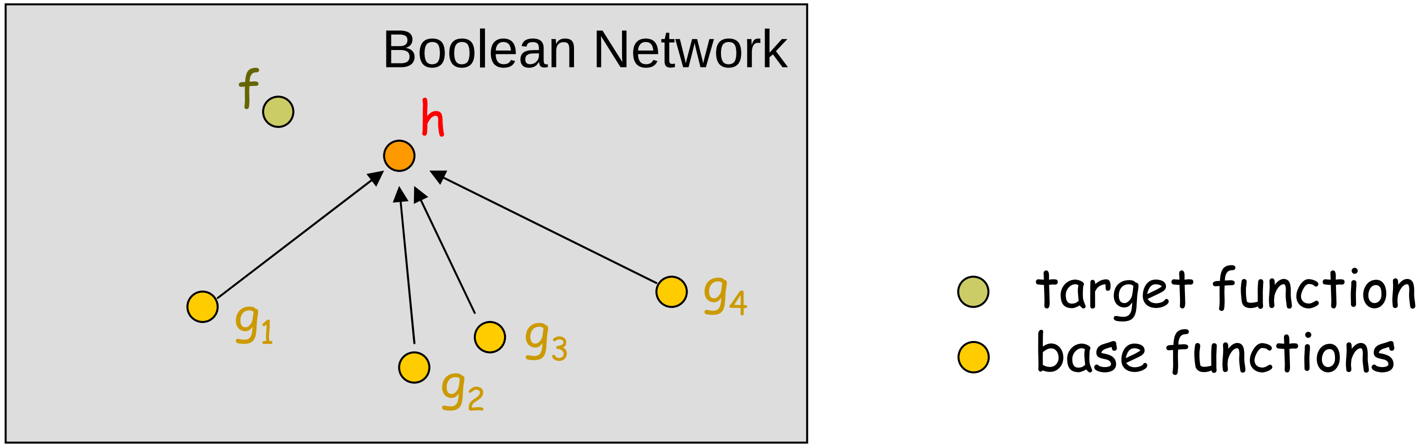
SAT Application
Functional Dependency

- $f(x)$ **functionally depends** on $g_1(x)$, $g_2(x)$, …, $g_m(x)$ if $f(x) = h(g_1(x), g_2(x), \ldots, g_m(x))$, denoted $h(G(x))$
  - Under what condition can function f be expressed as some function h over a set of given functions $G=\{g_1,\ldots,g_m\}$?
  - h exists $\Leftrightarrow$ $\nexists a,b$ such that $f(a)\neq f(b)$ and $G(a)=G(b)$

i.e., G is more distinguishing than f

SAT Application
Functional Dependency

- Applications of functional dependency
  - Resynthesis/rewiring
  - Redundant register removal
  - BDD minimization
  - Verification reduction
  - …

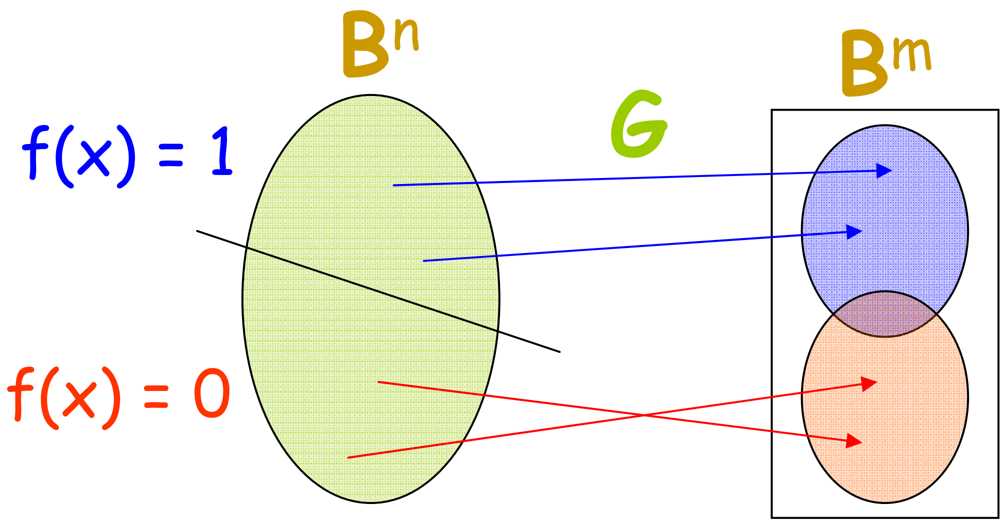


SAT Application
Functional Dependency

- Computing h

  $h^{on} = \{y \in \mathbf{B}^m : y = G(x) \text{ and } f(x) = 1,\ x \in \mathbf{B}^n\}$

  $h^{off} = \{y \in \mathbf{B}^m : y = G(x) \text{ and } f(x) = 0,\ x \in \mathbf{B}^n\}$



$h^{on} = \exists x.(y \equiv G) \wedge f$

$h^{off} = \exists x.(y \equiv G) \wedge \neg f$

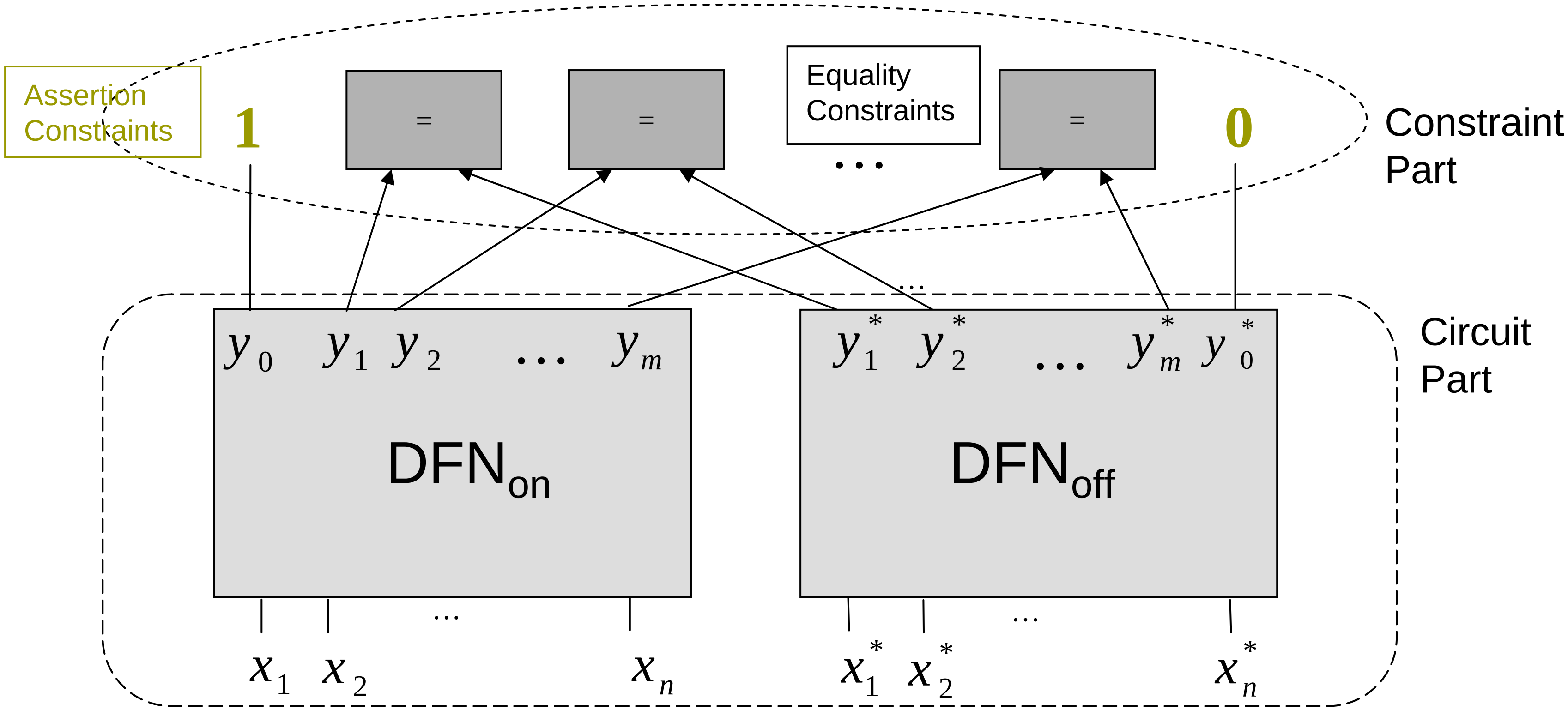
SAT Application
Functional Dependency

- h exists $\Leftrightarrow$
  $\nexists a,b$ such that $f(a) \neq f(b)$ and $G(a)=G(b)$,
  i.e., $(f(x) \not\equiv f(x^*)) \wedge (G(x) \equiv G(x^*))$ is UNSAT

- How to derive h? How to select G?

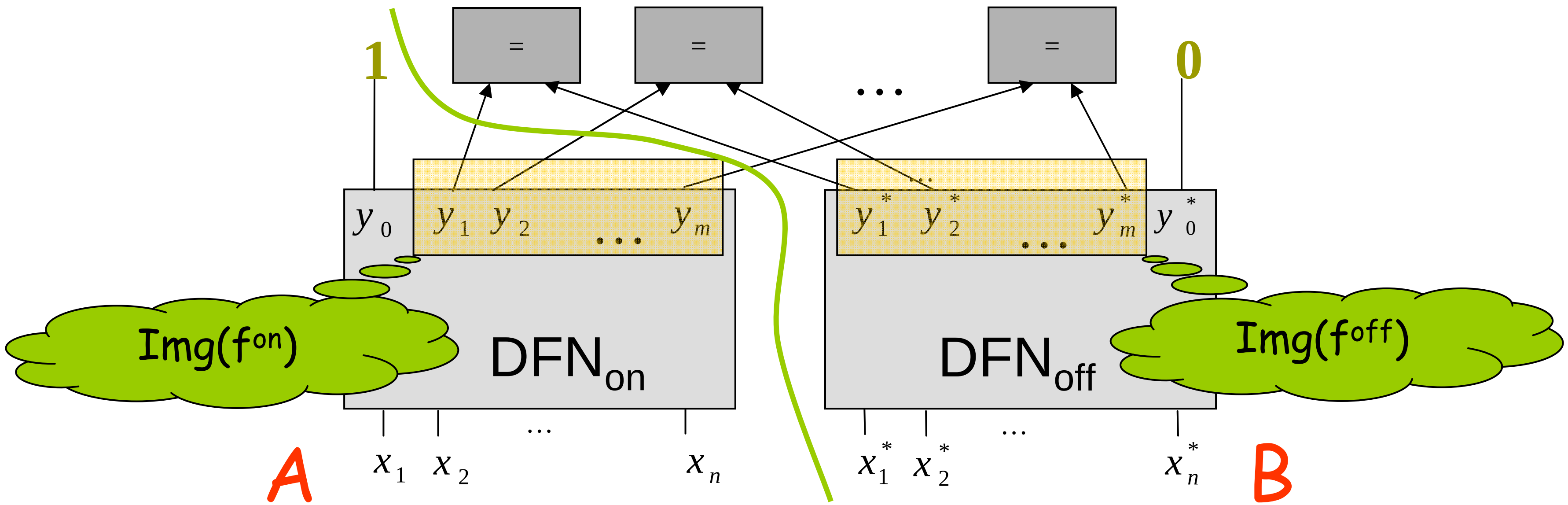
SAT Application
Functional Dependency

- $(f(x) \not\equiv f(x^*)) \wedge (G(x) \equiv G(x^*))$ is UNSAT



SAT Application
Functional Dependency

- Clause set *A*: C_{DFNon}, y_0
- Clause set B: C_{DFNoff}, $\neg y_0^*$, $(y_i \equiv y_i^*)$ for i =1,…,m
- I is an overapproximation of Img(f^{on}) and is disjoint from Img(f^{off})
- I only refers to $y_1, \ldots, y_m$
- Therefore, I corresponds to a feasible implementation of h



[Lee, J, Huang, Mishchenko, 2007]

Quantified Satisfiability

Quantified Boolean Formula

- A quantified Boolean formula (QBF) is often written in **prenex form** (with quantifiers placed on the left) as

$$\underbrace{Q_1\, x_1, \ldots, Q_n\, x_n}_{\text{prefix}}.\ \underbrace{\varphi}_{\text{matrix}}$$

for $Q_i \in \{\forall, \exists\}$ and φ a quantifier-free formula

  - If φ is further in CNF, the corresponding QBF is in the so-called **prenex CNF** (PCNF), the most popular QBF representation
  - Any QBF can be converted to PCNF

Quantified Boolean Formula

- Quantification order matters in a QBF
- A variable x_i in $(Q_1\ x_1, \ldots, Q_i\ x_i, \ldots, Q_n\ x_n.\ \varphi)$ is of **level** k if there are k quantifier alternations (i.e., changing from $\forall$ to $\exists$ or from $\exists$ to $\forall$) from Q_1 to Q_i.
  - Example

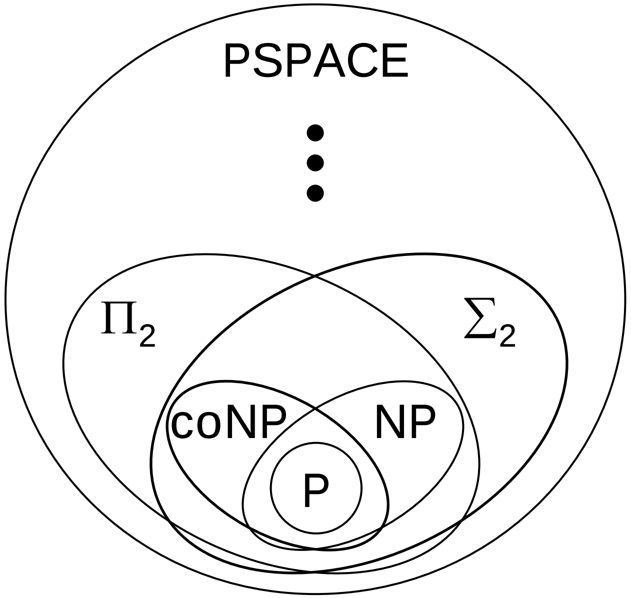
    $\forall a\ \exists b\ \forall c\ \forall d\ \exists e.\ \varphi$

    level(a)=0, level(b)=1, level(c)=2, level(d)=2, level(e)=3

Quantified Boolean Formula

- Many decision problems can be compactly encoded in QBFs

- In theory, QBF solving (QSAT) is PSPACE complete
  - The more the quantifier alternations, the higher the complexity in the Polynomial Hierarchy



- In practice, solvable QBFs are typically of size ~1,000 variables

QBF Solver

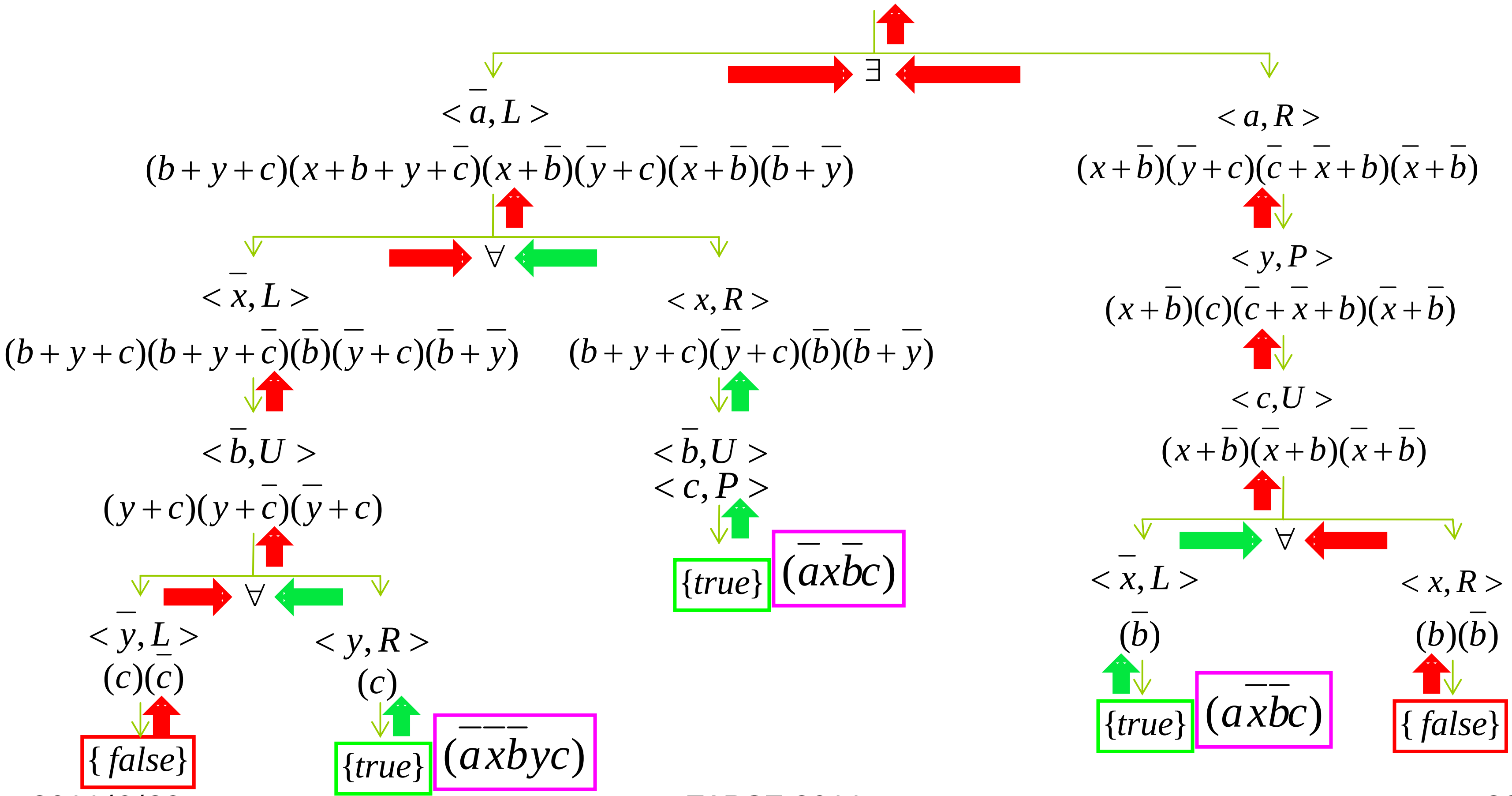
- QBF solver choices
  - Data structures for formula representation
    - **Prenex** vs. non-prenex
    - **Normal form** vs. non-normal form
      - CNF, NNF, BDD, AIG, etc.
  - Solving mechanisms
    - **Search**, Q-resolution, Skolemization, quantifier elimination, etc.
  - Preprocessing techniques

- Standard approach
  - Search-based PCNF formula solving (similar to SAT)
    - Both **clause learning** (from a conflicting assignment) and **cube learning** (from a satisfying assignment) are performed

QBF Solving

- Example

$$\exists a \forall x \exists b \forall y \exists c \quad (a+b+y+c)(a+x+b+y+\bar{c})(x+\bar{b})(\bar{y}+c)(\bar{c}+\bar{a}+\bar{x}+b)(\bar{x}+\bar{b})(a+\bar{b}+\bar{y})$$



Q-Resolution

- **Q-resolution** on PCNF is similar to resolution on CNF, except that the pivots are restricted to existentially quantified variables and the additional rule of ∀**-reduction**

$$\frac{C_1 \vee x \qquad\qquad C_2 \vee \neg x}{\forall\text{-RED}(C_1 \vee C_2)}$$

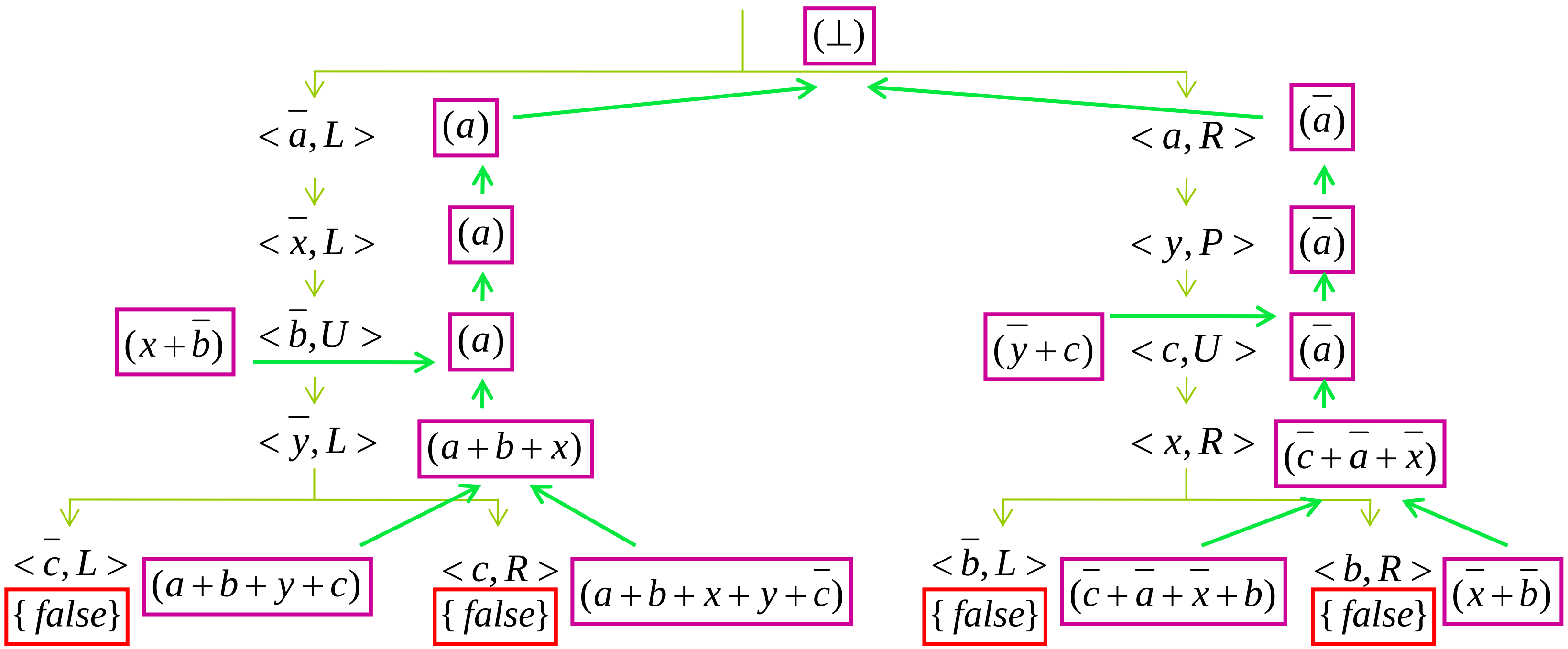
  where operator ∀-RED removes from $C_1 \vee C_2$ the universally (∀) quantified variables whose quantification levels are greater than any of the existentially (∃) quantified variables in $C_1 \vee C_2$
  - E.g.,
    prefix: $\forall a\ \exists b\ \forall c\ \forall d\ \exists e$
    $\forall$-RED(a+b+c+d) = (a+b)

- Q-resolution is complete for QBF solving
  - A PCNF formula is unsatisfiable if and only if there exists a Q-resolution sequence leading to the empty clause

Q-Resolution

Example (cont'd)

$$\exists a \forall x \exists b \forall y \exists c \quad (a+b+y+c)(a+x+b+y+\bar{c})(x+\bar{b})(\bar{y}+c)(\bar{c}+\bar{a}+\bar{x}+b)(\bar{x}+\bar{b})(a+\bar{b}+\bar{y})$$



Skolemization

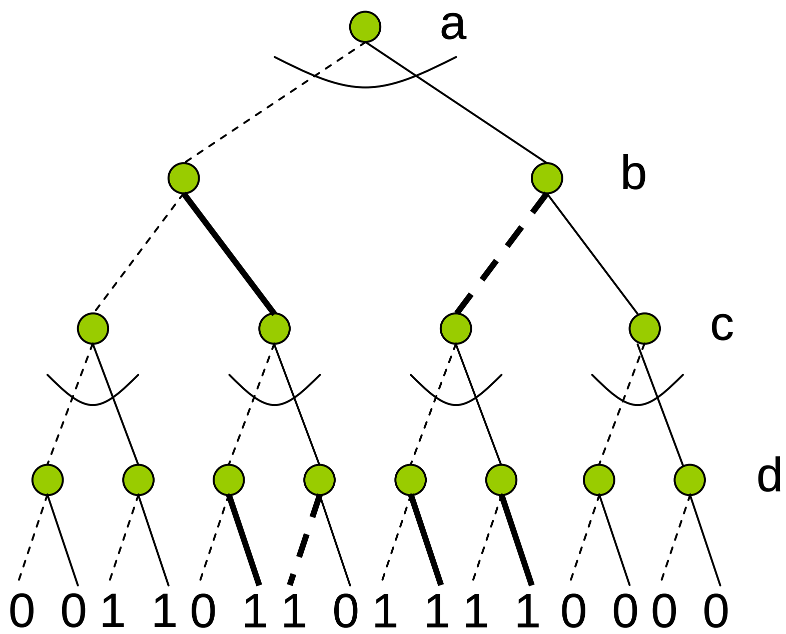
- Skolemization and Skolem normal form
  - Existentially quantified variables are replaced with function symbols
  - QBF prefix contains only two quantification levels
    - $\exists$ function symbols, $\forall$ variables

- Example

$\forall a\ \exists b\ \forall c\ \exists d.$
$(\neg a+\neg b)(\neg b+\neg c+\neg d)(\neg b+c+d)(a+b+c)$

Skolem functions

$\exists F_b(a)\ \exists F_d(a,c)\ \forall a\ \forall c.$
$(\neg a+\neg F_b)(\neg F_b+\neg c+\neg F_d)(\neg F_b+c+F_d)(a+F_b+c)$



QBF Certification

- QBF certification
  - Ensure correctness and, more importantly, provide useful information
  - Certificates
    - True QBF: term-resolution proof / Skolem-function (SF) model
      - SF model is more useful in practical applications
    - False QBF: clause-resolution proof / Herbrand-function (HF) countermodel
      - HF countermodel is more useful in practical applications

- Solvers and certificates
  - To date, only Skolemization-based solvers (e.g., sKizzo, squolem, Ebddres) can provide SFs
  - Search-based solvers (e.g., QuBE) are the most popular and can be instrumented to provide resolution proofs

QBF Certification

- Solvers and certificates

Solver	Algorithm	Certificate	
		True QBF	False QBF
QuBE-cert	search	Cube resolution	Clause resolution
yQuaffle	search	Cube resolution	Clause resolution
Ebddres	Skolemization	Skolem function	Clause resolution
sKizzo	Skolemization	Skolem function	-
squolem	Skolemization	Skolem function	Clause resolution

QBF Certification

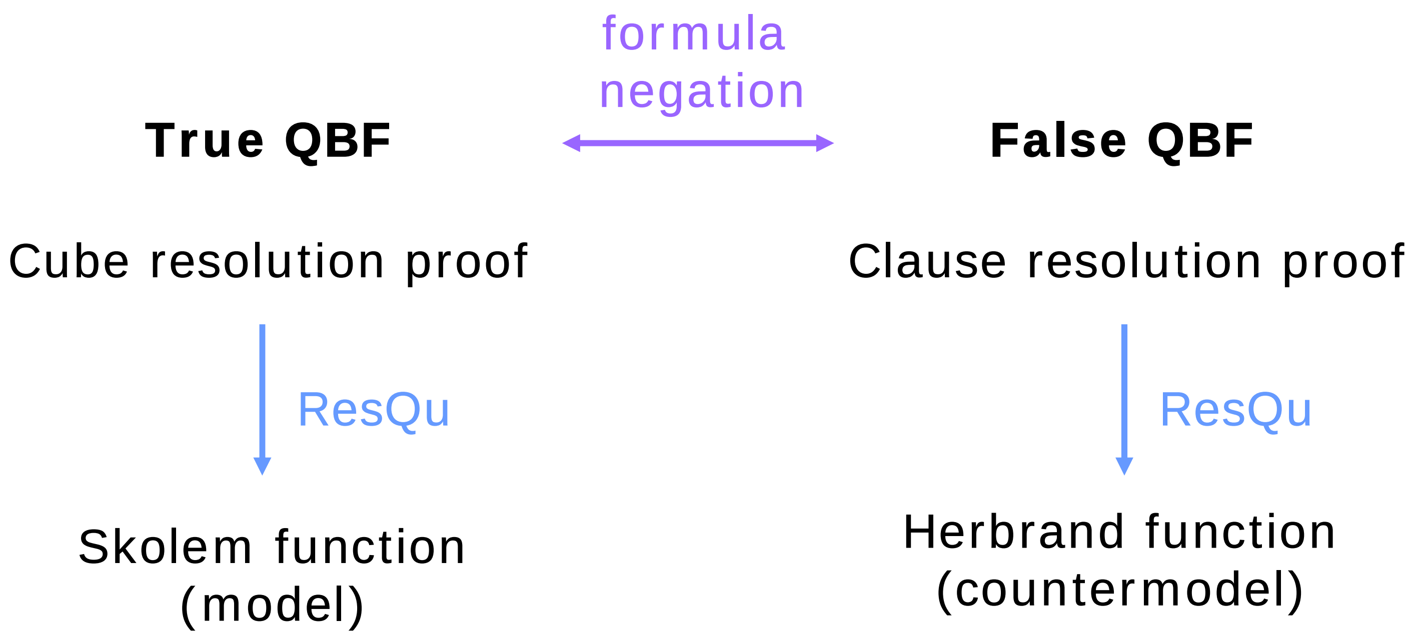
- Incomplete picture of QBF certification

	Syntactic Certificate	Semantic Certificate
True QBF	Cube-resolution proof	Skolem-function model
False QBF	Clause-resolution proof	**?**

- Recent progress
  - Herbrand-function countermodel
    - [Balabanov, J, 2011 (ResQu)]
  - Syntactic to semantic certificate conversion
    - Linear time [Balabanov, J, 2011 (ResQu)]

QBF Certification

- Unified QBF certification



ResQu

- A Skolem-function model (Herbrand-function countermodel) for a true (false) QBF can be derived from its cube (clause) resolution proof

- A **Right-First-And-Or (RFAO) formula** is recursively defined as follows.

  φ := clause | cube | clause $\wedge$ φ | cube $\vee$ φ

  - E.g.,

    (a'+b) $\wedge$ ac $\vee$ (b'+c') $\wedge$ bc

    = ((a'+b) $\wedge$ (ac $\vee$ ((b'+c') $\wedge$ bc)))

ResQu

Countermodel_construct

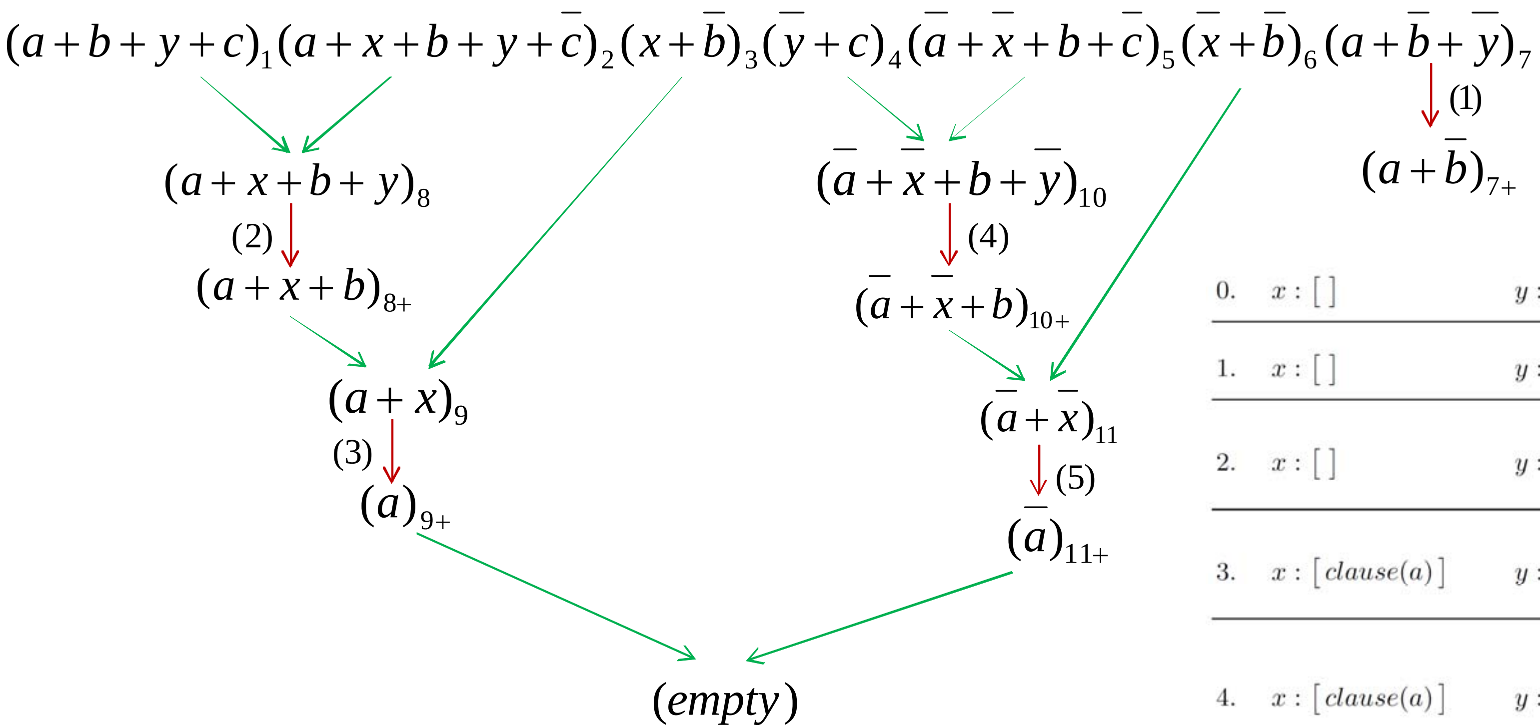
input: a false QBF Φ and its clause-resolution DAG $G_\Pi(V_\Pi, E_\Pi)$
output: a countermodel in RFAO formulas
begin

- **foreach** universal variable x of Φ
  - `RFAO_node_array`$[x] := \emptyset$;
- **foreach** vertex v of G_Π in topological order
  - **if** $v.clause$ resulted from $\forall$-reduction on $u.clause$, i.e., $(u, v) \in E_\Pi$
    - $v.cube := \neg(v.clause)$;
    - **foreach** universal variable x reduced from $u.clause$ to get $v.clause$
      - **if** x appears as positive literal in $u.clause$
        - **push** $v.clause$ to `RFAO_node_array`$[x]$;
      - **else if** x appears as negative literal in $u.clause$
        - **push** $v.cube$ to `RFAO_node_array`$[x]$;
  - **if** $v.clause$ is the empty clause
    - **foreach** universal variable x of Φ
      - simplify `RFAO_node_array`$[x]$;
    - **return** `RFAO_node_array`'s;

end

ResQu

- Example
  - ∃a∀x∃b∀y∃c



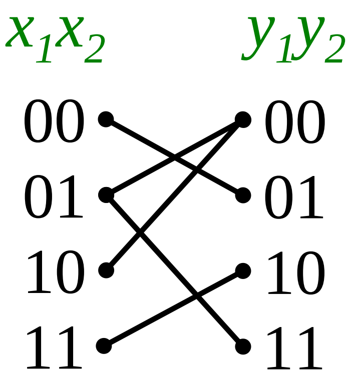
0.	$x : [\,]$	$y : [\,]$
1.	$x : [\,]$	$y : [cube(\bar{a}b)]$
2.	$x : [\,]$	$y : [cube(\bar{a}b), clause(a+x+b)]$
3.	$x : [clause(a)]$	$y : [cube(\bar{a}b), clause(a+x+b)]$
4.	$x : [clause(a)]$	$y : [cube(\bar{a}b), clause(a+x+b), cube(ax\bar{b})]$
5.	$x : [clause(a), cube(a)]$	$y : [cube(\bar{a}b), clause(a+x+b), cube(ax\bar{b})]$

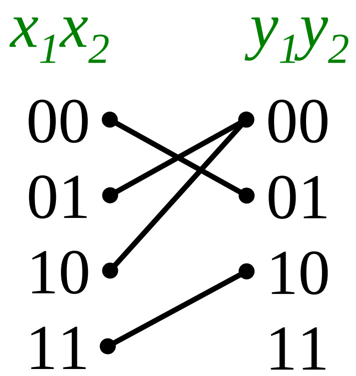
QBF Certification

- Applications of Skolem/Herbrand functions
  - Program synthesis
  - Winning strategy synthesis in two player games
  - Plan derivation in AI
  - Logic synthesis
  - …

QBF Application
Relation Determinization

- Relation *R(X, Y)*
  - Allow one-to-many mappings
    - Can describe non-deterministic behavior
  - More generic than functions
- Function *F(X)*
  - Disallow one-to-many mappings
    - Can only describe deterministic behavior
  - A special case of relation



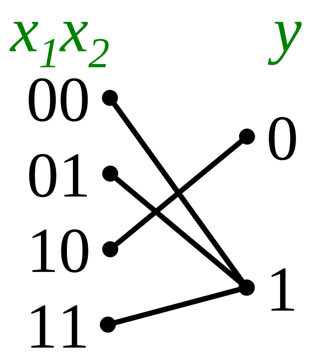


$f_1 = x_1\, x_2$

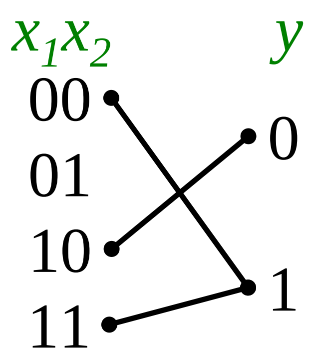
$f_2 = \neg x_1 \neg x_2$

QBF Application
Relation Determinization

- Total relation
  - Every input element is mapped to at least one output element

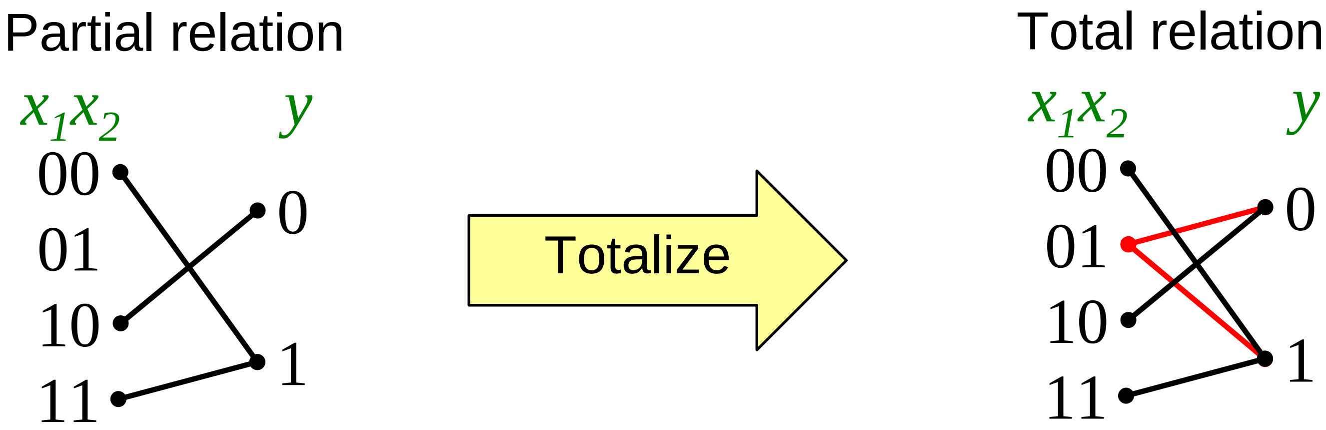


- Partial relation
  - Some input element is not mapped to any output element



QBF Application
Relation Determinization

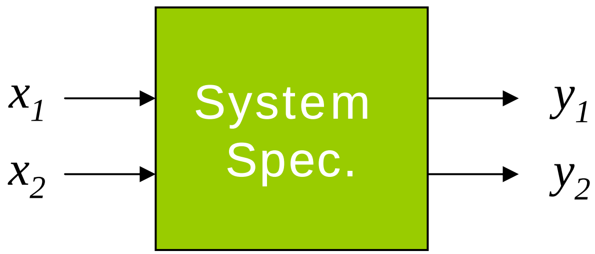
- A partial relation can be **totalized**
  - Assume that the input element not mapped to any output element is a don't care

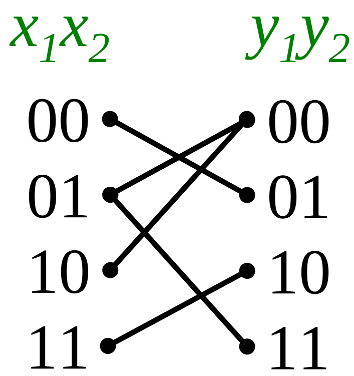


$$T(X, y) = R(X, y) \vee \forall y.\, \neg R(X, y)$$

QBF Application
Relation Determinization

- Applications of Boolean relation
  - In high-level design, Boolean relations can be used to describe (nondeterministic) specifications
  - In gate-level design, Boolean relations can be used to characterize the flexibility of sub-circuits
    - Boolean relations are more powerful than traditional don't-care representations



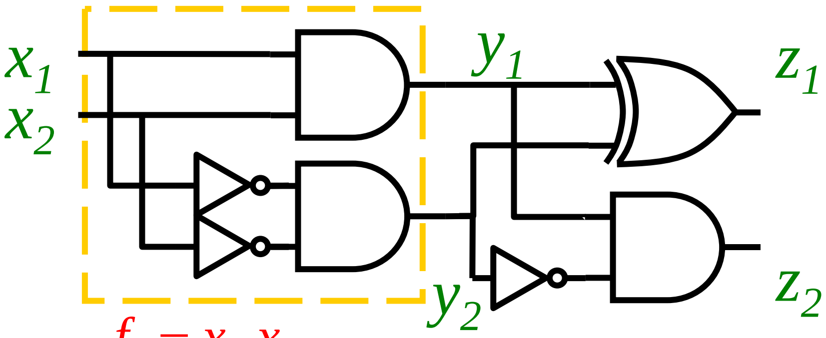


QBF Application
Relation Determinization

- Relation determinization
  - For hardware implementation of a system, we need functions rather than relations
    - Hardware systems are intrinsically deterministic
      - One input stimulus results in one output response
  - To simplify implementation, we can explore the flexibilities described by a relation for optimization

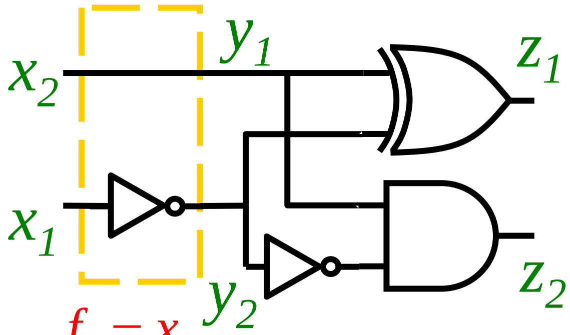
QBF Application
Relation Determinization

Example



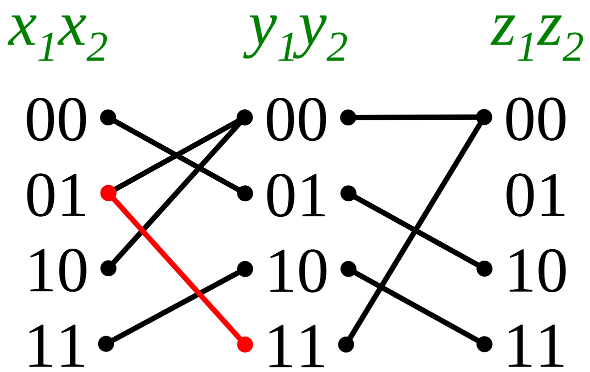
$f_1 = x_1\, x_2$

$f_2 = \neg x_1 \neg x_2$



$f_1 = x_2$

$f_2 = \neg x_1$



QBF Application
Relation Determinization

- Given a *nondeterministic* Boolean relation $R(X, Y)$, how to determinize and extract functions from it?

- Solve QBF

  $$\forall x_1,\ldots,\forall x_m, \exists y_1,\ldots, \exists y_n. R(x_1,\ldots,x_m, y_1, \ldots, y_n)$$

  - The Skolem functions of variables $y_1, \ldots, y_n$ correspond to the output-functions we want

QBF Application
Program Synthesis

- Program synthesis by sketching
  - [Solar-Lezama et al., 2006]

- Example

```
Spec:
int foo (int x){
    return x+x;
}
```

```
Sketch:
int bar (int x) implements foo{
    return x << ??;
}
```

```
Result:
int bar (int x) implements foo{
    return x << 1;
}
```

QBF Application
Program Synthesis

- Sketch synthesis can be solved by searching for control values satisfying

$$\exists c\ \forall x.\ Spec(x) = Sk(x,c)$$

  - We are interested to derive the Skolem function (in this case, constant) of c

Conclusions

- Modern SAT/QSAT solvers are powerful tools for solving large-scale synthesis, verification, and other computer science problems

- Certificates of SAT/QSAT solving may be utilized to extract essential information for applications in synthesis and verification

- Understanding how solvers work helps practitioners formulate and solve real-world problems

Suggested Further Exploration

- SMT solvers and their applications in program analysis and verification

Contributors

- Valeriy Balabanov, NTU
- Wei-Lun Hung, NTU
- Chih-Chun Lee, NTU
- Hsuan-Po Lin, NTU
- Alan Mishchenko, UC Berkeley

Thank You!

- Questions?