Boolean Satisfiability and Its Applications to Synthesis & Verification

Jie-Hong Roland Jiang (江介宏)

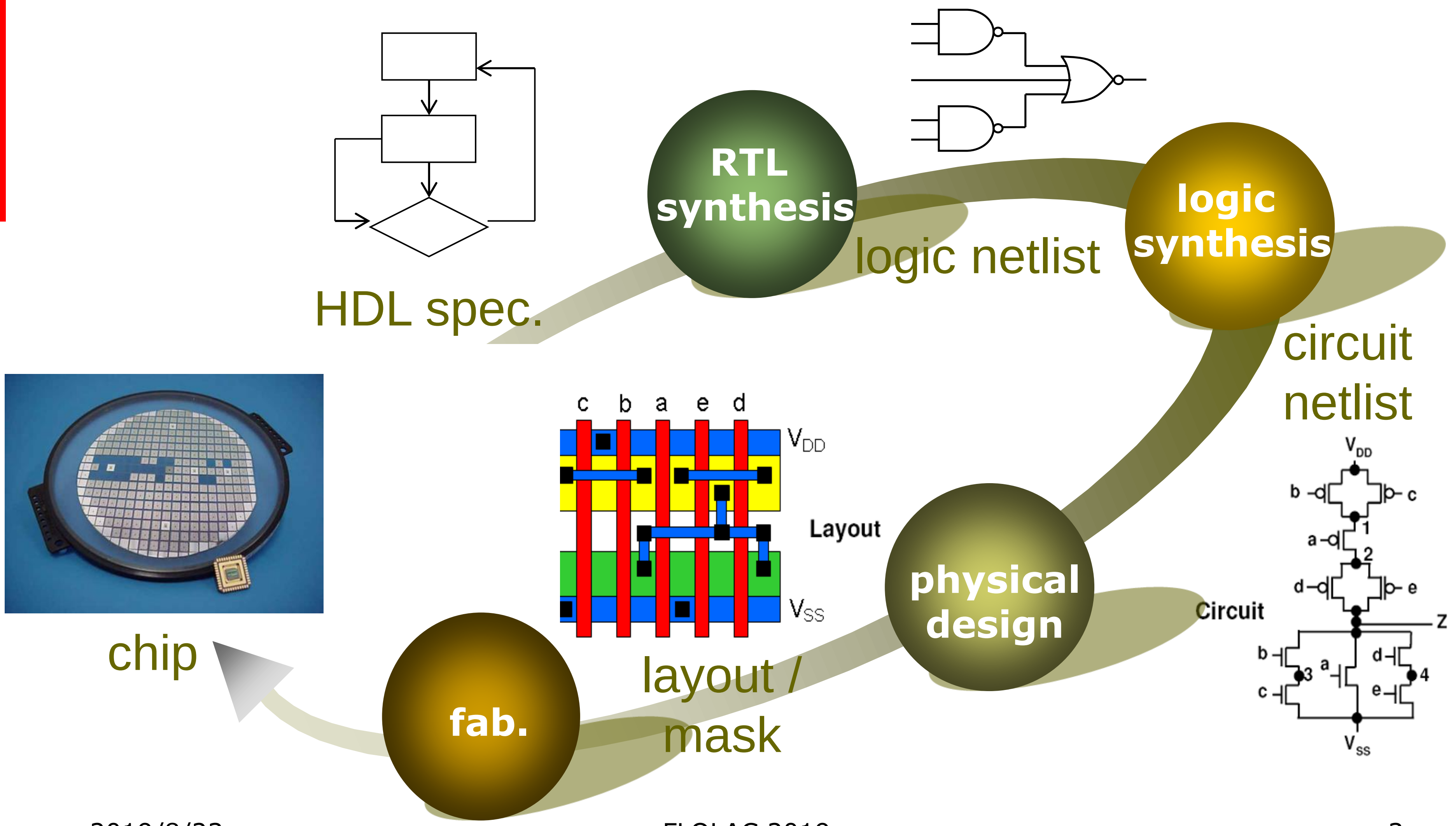
ALCom Lab

Department of Electrical Engineering,
Graduate Institute of Electronics Engineering
National Taiwan University

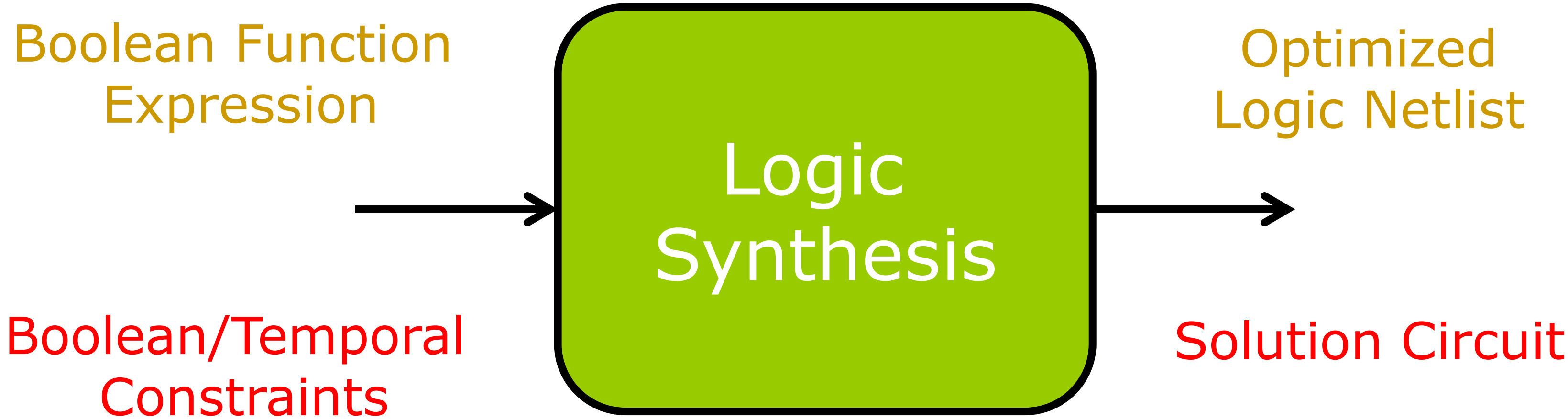
Outline

- Logic synthesis & verification
- Boolean function representation
- Propositional satisfiability & applications
- Quantified Boolean satisfiability & applications
- Stochastic Boolean satisfiability & applications

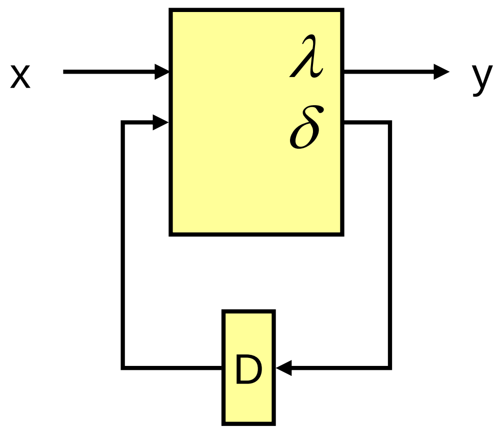
IC Design Flow



Logic Synthesis

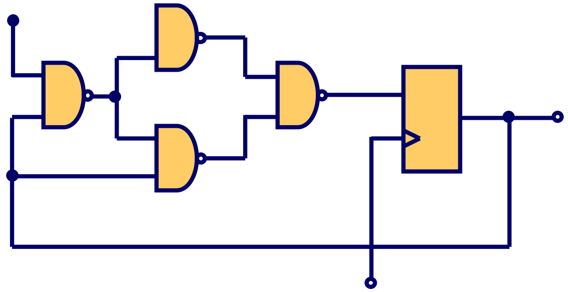


Logic Synthesis



Given: Functional description of finite-state machine F(Q,X,Y,δ,λ) where:

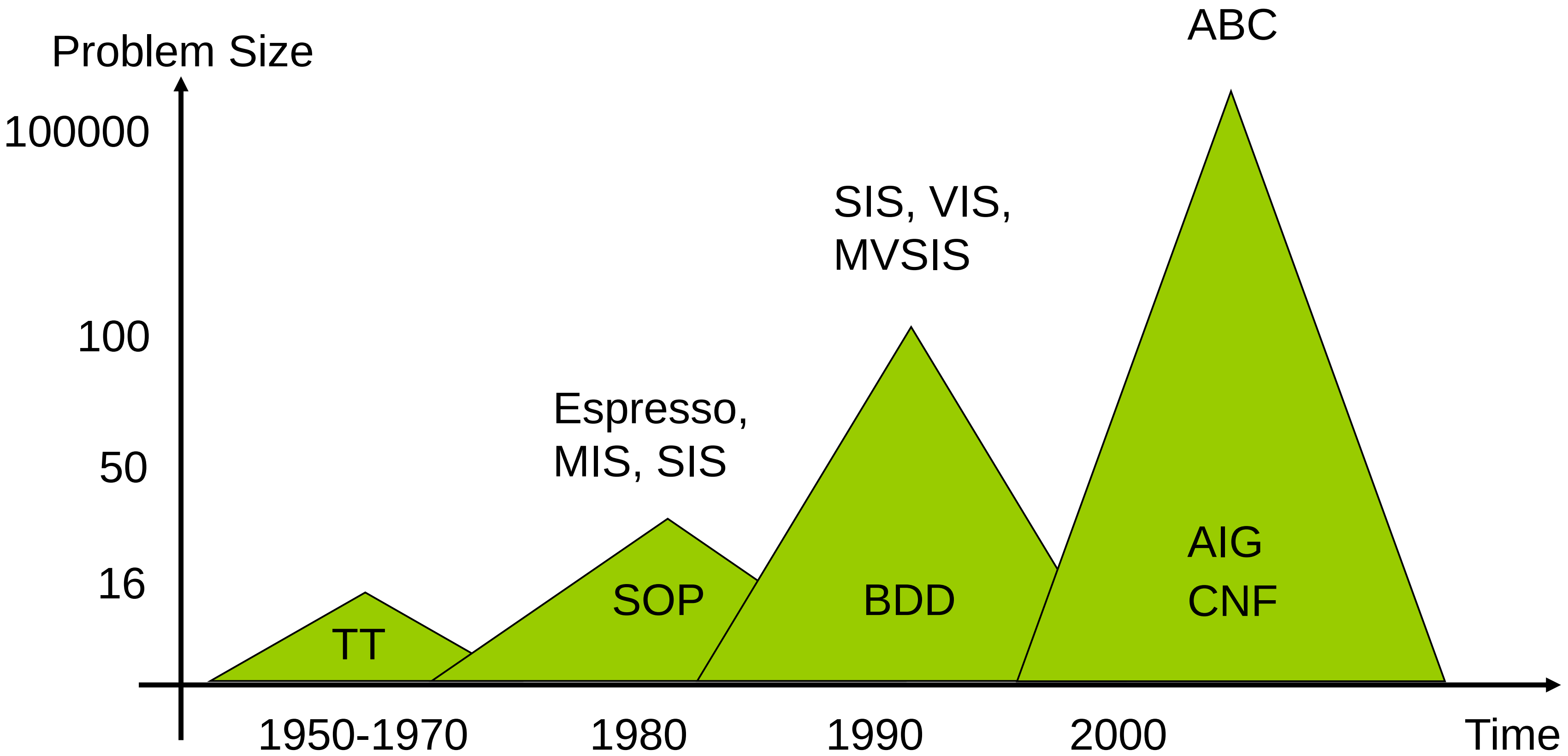
Q: Set of internal states
X: Input alphabet
Y: Output alphabet
δ: X x Q $\rightarrow$ Q (next state *function*)
λ: X x Q $\rightarrow$ Y (output *function*)



Target: Circuit C(G, W) where:
G: set of circuit components g $\in$ {gates, FFs, etc.}
W: set of wires connecting G

Backgrounds

- Historic evolution of data structures and tools in logic synthesis and verification



Courtesy of Alan Mishchenko

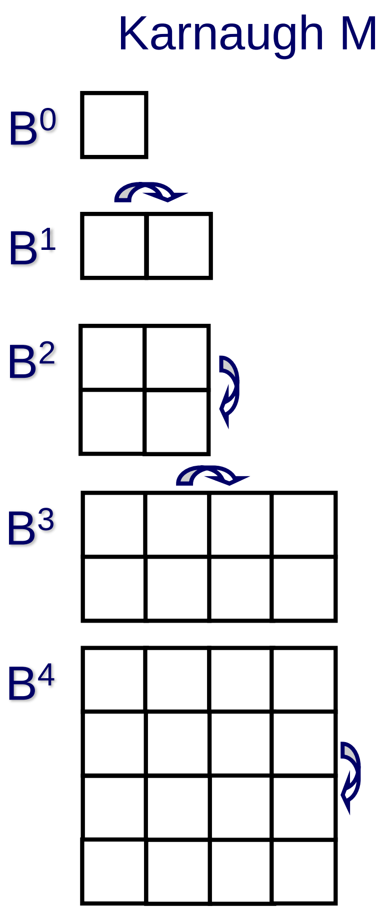
Boolean Function Representation

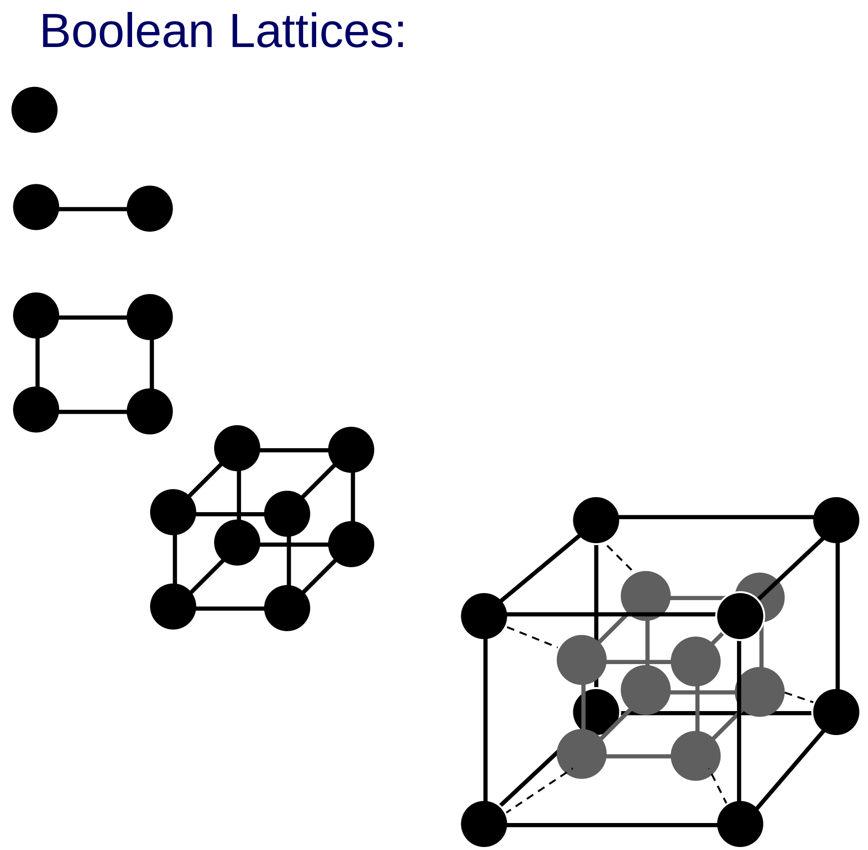
- Logic synthesis translates Boolean functions into circuits

- We need representations of Boolean functions for two reasons:
  - to represent and manipulate the actual circuit that we are implementing
  - to facilitate *Boolean reasoning*

Boolean Space

- $B = \{0,1\}$
- $B^2 = \{0,1\}\times\{0,1\} = \{00, 01, 10, 11\}$





Boolean Function

- A Boolean function f over input variables: $x_1, x_2, \ldots, x_m$, is a mapping f: $\mathbf{B}^m \rightarrow Y$, where $\mathbf{B} = \{0,1\}$ and $Y = \{0,1,d\}$
  - E.g.
  - The output value of $f(x_1, x_2, x_3)$, say, partitions $\mathbf{B}^m$ into three sets:
    - on-set ($f = 1$)
      - E.g. {010, 011, 110, 111} (characteristic function $f^1 = x_2$)
    - off-set ($f = 0$)
      - E.g. {100, 101} (characteristic function $f^0 = x_1 \neg x_2$)
    - don't-care set ($f = d$)
      - E.g. {000, 001} (characteristic function $f^d = \neg x_1 \neg x_2$)

- f is an incompletely specified function if the don't-care set is nonempty. Otherwise, f is a completely specified function
  - Unless otherwise said, a Boolean function is meant to be completely specified

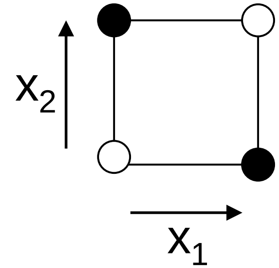
Boolean Function

- A Boolean function f: $\mathbf{B}^n \rightarrow \mathbf{B}$ over variables $x_1,\ldots,x_n$ maps each Boolean valuation (truth assignment) in $\mathbf{B}^n$ to 0 or 1

Example

$f(x_1,x_2)$ with $f(0,0) = 0$, $f(0,1) = 1$, $f(1,0) = 1$, $f(1,1) = 0$

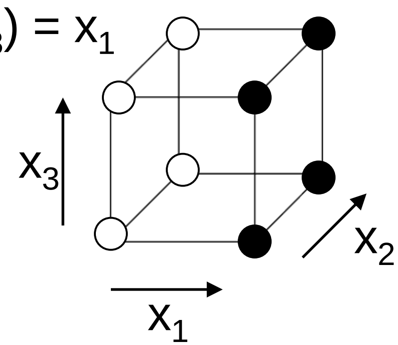
		x_2
	0	1
x_1	1	0



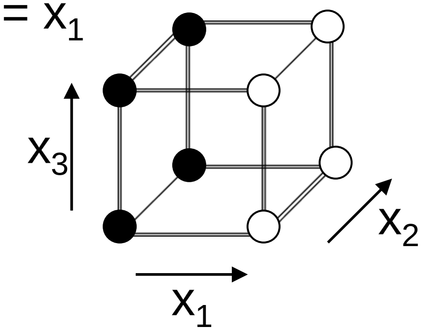
Boolean Function

- Onset of f, denoted as f^1, is $f^1 = \{v \in \mathbf{B}^n \mid f(v)=1\}$
  - If $f^1 = \mathbf{B}^n$, f is a tautology
- Offset of f, denoted as f^0, is $f^0 = \{v \in \mathbf{B}^n \mid f(v)=0\}$
  - If $f^0 = \mathbf{B}^n$, f is unsatisfiable. Otherwise, f is satisfiable.
- f^1 and f^0 are sets, not functions!
- Boolean functions f and g are equivalent if $\forall v \in \mathbf{B}^n.\ f(v) = g(v)$ where v is a truth assignment or Boolean valuation
- A literal is a Boolean variable x or its negation x' (or $\bar{x}$, $\neg x$) in a Boolean formula

$f(x_1, x_2, x_3) = x_1$

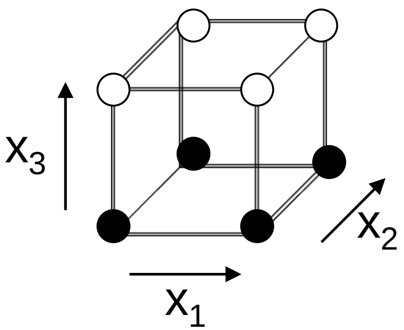


$f(x_1, x_2, x_3) = \overline{x_1}$



Boolean Function

- There are 2^n vertices in $\mathbf{B}^n$
- There are 2^{2^n} distinct Boolean functions
  - Each subset $f^1 \subseteq \mathbf{B}^n$ of vertices in $\mathbf{B}^n$ forms a distinct Boolean function f with onset f^1



$x_1x_2x_3$		f
0 0 0		1
0 0 1		0
0 1 0		1
0 1 1		0
1 0 0	$\Rightarrow$	1
1 0 1		0
1 1 0		1
1 1 1		0

Boolean Operations

Given two Boolean functions:

$f : \mathbf{B}^n \to \mathbf{B}$

$g : \mathbf{B}^n \to \mathbf{B}$

- $h = f \wedge g$ from AND operation is defined as
  $h^1 = f^1 \cap g^1$; $h^0 = \mathbf{B}^n \setminus h^1$
- $h = f \vee g$ from OR operation is defined as
  $h^1 = f^1 \cup g^1$; $h^0 = \mathbf{B}^n \setminus h^1$
- $h = \neg f$ from COMPLEMENT operation is defined as
  $h^1 = f^0$; $h^0 = f^1$

Cofactor and Quantification

Given a Boolean function:

$f : \mathbf{B}^n \rightarrow \mathbf{B}$, with the input variable $(x_1,x_2,\ldots,x_i,\ldots,x_n)$

- Positive cofactor on variable x_i
  $h = f_{x_i}$ is defined as $h = f(x_1,x_2,\ldots,1,\ldots,x_n)$
- Negative cofactor on variable x_i
  $h = f_{\neg x_i}$ is defined as $h = f(x_1,x_2,\ldots,0,\ldots,x_n)$
- Existential quantification over variable x_i
  $h = \exists x_i.\ f$ is defined as $h = f(x_1,x_2,\ldots,0,\ldots,x_n) \vee f(x_1,x_2,\ldots,1,\ldots,x_n)$
- Universal quantification over variable x_i
  $h = \forall x_i.\ f$ is defined as $h = f(x_1,x_2,\ldots,0,\ldots,x_n) \wedge f(x_1,x_2,\ldots,1,\ldots,x_n)$
- Boolean difference over variable x_i
  $h = \partial f/\partial x_i$ is defined as $h = f(x_1,x_2,\ldots,0,\ldots,x_n) \oplus f(x_1,x_2,\ldots,1,\ldots,x_n)$

Boolean Function Representation

- Some common representations:
  - Truth table
  - Boolean formula
    - SOP (sum-of-products, or called disjunctive normal form, DNF)
    - POS (product-of-sums, or called conjunctive normal form, CNF)
  - BDD (binary decision diagram)
  - Boolean network (consists of nodes and wires)
    - Generic Boolean network
      - Network of nodes with generic functional representations or even subcircuits
    - Specialized Boolean network
      - Network of nodes with SOPs (PLAs)
      - And-Inv Graph (AIG)

- Why different representations?
  - Different representations have their own strengths and weaknesses (no single data structure is best for all applications)

Boolean Function Representation
Truth Table

- Truth table (function table for multi-valued functions):
  The truth table of a function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ is a tabulation of its value at each of the 2^n vertices of $\mathbf{B}^n$.

  In other words the truth table lists all mintems

  Example: $f = a'b'c'd + a'b'cd + a'bc'd + ab'c'd + ab'cd + abc'd + abcd' + abcd$

  The truth table representation is
  - impractical for large n
  - canonical

  If two functions are the equal, then their canonical representations are isomorphic.

	abcd	f
0	0000	0
1	0001	1
2	0010	0
3	0011	1
4	0100	0
5	0101	1
6	0110	0
7	0111	0

	abcd	f
8	1000	0
9	1001	1
10	1010	0
11	1011	1
12	1100	0
13	1101	1
14	1110	1
15	1111	1

Boolean Function Representation
Boolean Formula

- A Boolean formula is defined inductively as an expression with the following formation rules (syntax):

formula ::=		'(' formula ')'	
	|	Boolean constant	(**true** or **false**)
	|	<Boolean variable>	
	|	formula "+" formula	(OR operator)
	|	formula "·" formula	(AND operator)
	|	$\neg$ formula	(complement)

Example

$f = (x_1 \cdot x_2) + (x_3) + \neg(\neg(x_4 \cdot (\neg x_1)))$

typically "·" is omitted and '(', ')' are omitted when the operator priority is clear, e.g., $f = x_1\, x_2 + x_3 + x_4 \neg x_1$

Boolean Function Representation
Boolean Formula in SOP

- Any function can be represented as a sum-of-products (SOP), also called sum-of-cubes (a cube is a product term), or disjunctive normal form (DNF)

  Example
  $\varphi = ab + a'c + bc$

Boolean Function Representation
Boolean Formula in POS

- Any function can be represented as a product-of-sums (POS), also called conjunctive normal form (CNF)
  - Dual of the SOP representation

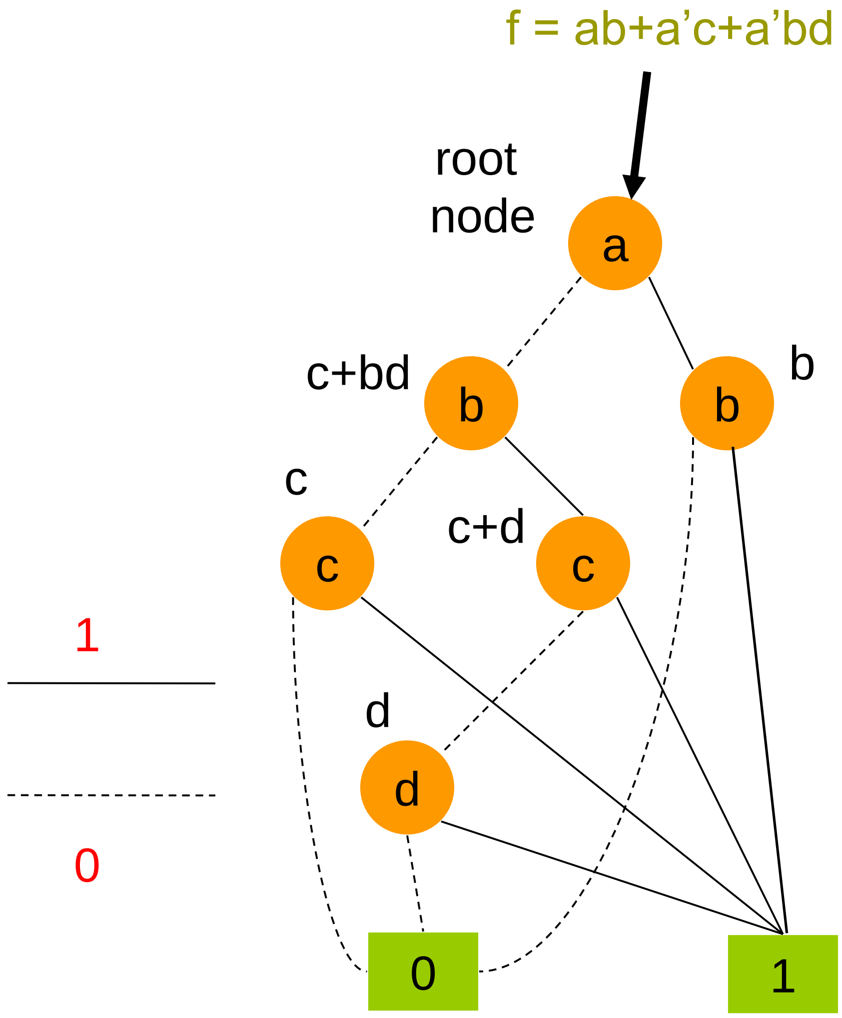
Example

- $\varphi = (a+b'+c)\,(a'+b+c)\,(a+b'+c')\,(a+b+c)$

- Exercise: Any Boolean function in POS can be converted to SOP using De Morgan's law and the distributive law, and vice versa

Boolean Function Representation
Binary Decision Diagram

- BDD – a graph representation of Boolean functions
  - A leaf node represents constant 0 or 1
  - A non-leaf node represents a decision node (multiplexer) controlled by some variable
  - Can make a BDD representation **canonical** by imposing the variable ordering and reduction criteria (ROBDD)



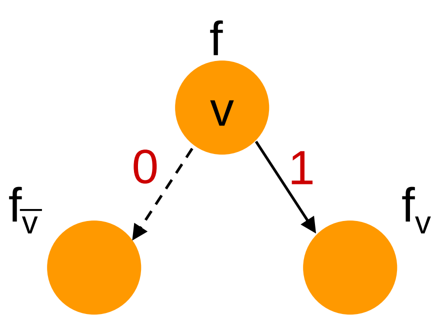
Boolean Function Representation
Binary Decision Diagram

- Any Boolean function *f* can be written in term of **Shannon expansion**

$$f = v\, f_v + \neg v\, f_{\neg v}$$

  - Positive cofactor: $f_{xi} = f(x_1,\ldots,x_i=1,\ldots, x_n)$
  - Negative cofactor: $f_{\neg xi} = f(x_1,\ldots,x_i=0,\ldots, x_n)$

- BDD is a compressed Shannon cofactor tree:
  - The two children of a node with function *f* controlled by variable *v* represent two sub-functions f_v and $f_{\neg v}$

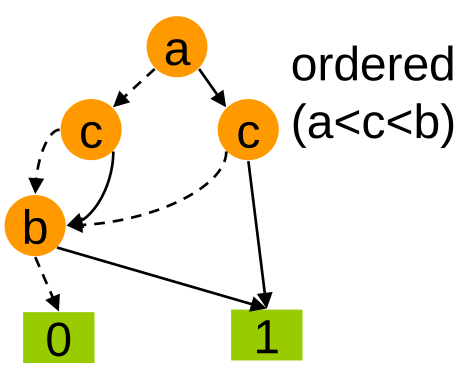


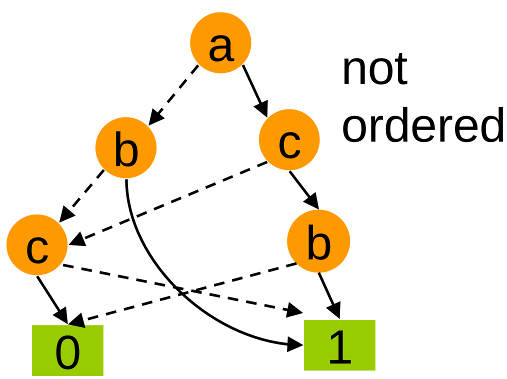
Boolean Function Representation
Binary Decision Diagram

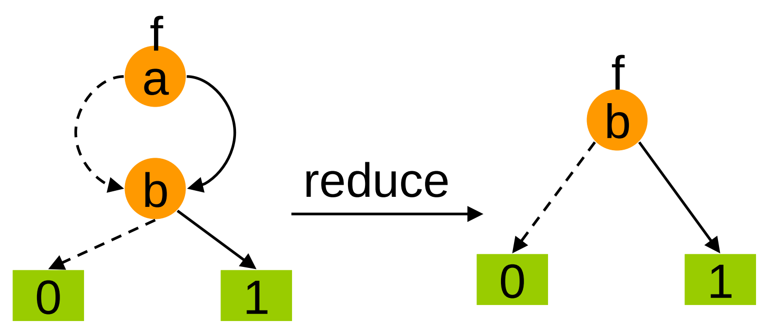
- Reduced and ordered BDD (ROBDD) is a **canonical** Boolean function representation
  - Ordered:
    - cofactor variables are in the same order along all paths

      $x_{i_1} < x_{i_2} < x_{i_3} < \ldots < x_{i_n}$
  - Reduced:
    - any node with two identical children is removed
    - two nodes with isomorphic BDD's are merged

      These two rules make any node in an ROBDD represent a distinct logic function

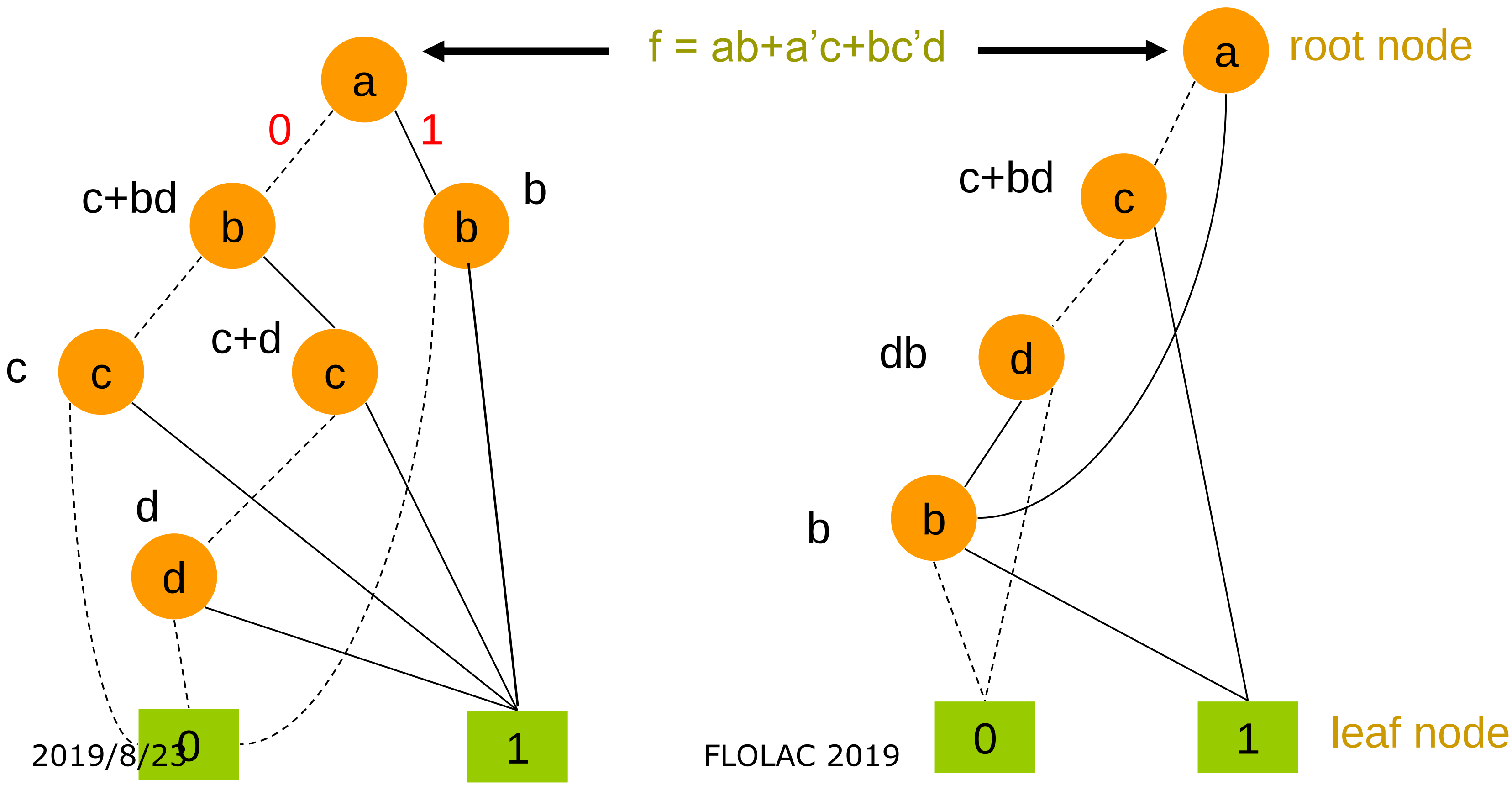






Boolean Function Representation
Binary Decision Diagram

- For a Boolean function,
  - ROBDD is unique with respect to a given variable ordering
  - Different orderings may result in different ROBDD structures



Boolean Function Representation
Boolean Network

- A Boolean network is a directed graph $C(G,N)$ where G are the gates and $N \subseteq (G \times G)$ are the directed edges (nets) connecting the gates.

  Some of the vertices are designated:
  Inputs: $I \subseteq G$
  Outputs: $O \subseteq G$
  $I \cap O = \emptyset$

  Each gate g is assigned a Boolean function f_g which computes the output of the gate in terms of its inputs.

Boolean Function Representation
Boolean Network

- The fanin FI(g) of a gate g are the predecessor gates of g:
  $FI(g) = \{g' \mid (g',g) \in N\}$ (N: the set of nets)

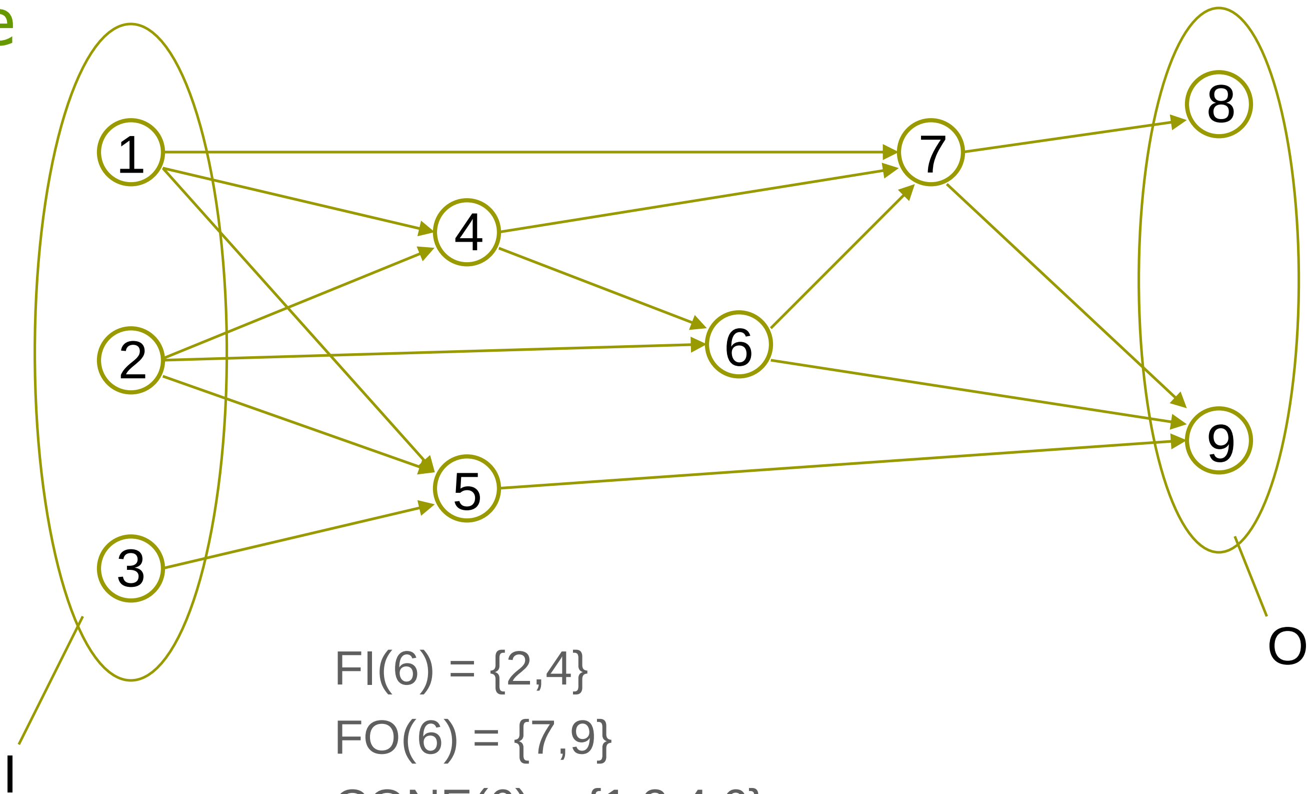
- The fanout FO(g) of a gate g are the successor gates of g:
  $FO(g) = \{g' \mid (g,g') \in N\}$

- The cone CONE(g) of a gate g is the transitive fanin (TFI) of g and g itself

- The support SUPPORT(g) of a gate g are all inputs in its cone:
  $SUPPORT(g) = CONE(g) \cap I$

Boolean Function Representation
Boolean Network

Example



FI(6) = {2,4}

FO(6) = {7,9}

CONE(6) = {1,2,4,6}

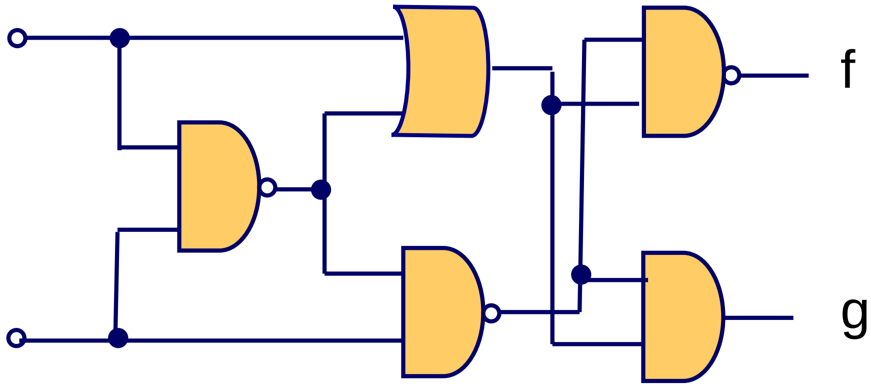
SUPPORT(6) = {1,2}

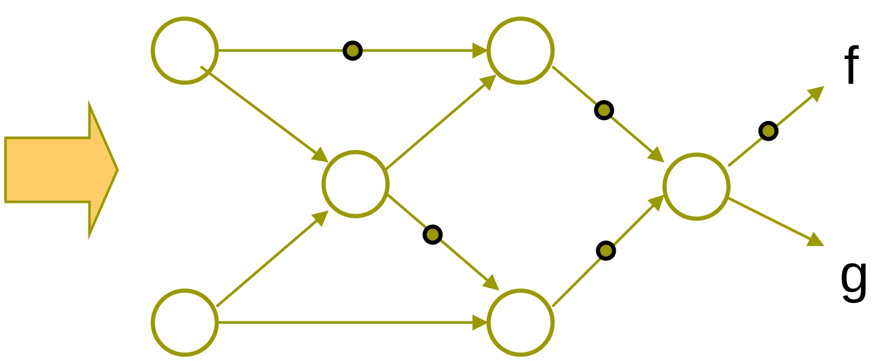
Every node may have its own function

Boolean Function Representation
And-Inverter Graph

- AND-INVERTER graphs (AIGs)
  vertices: 2-input AND gates
  edges: interconnects with (optional) dots representing INVs

- Hash table to identify and reuse structurally isomorphic circuits



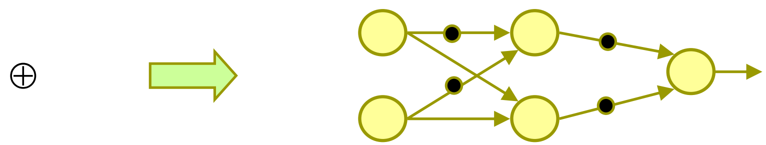


Boolean Function Representation

- Truth table
  - Canonical
  - Useful in representing small functions
- SOP
  - Useful in two-level logic optimization, and in representing local node functions in a Boolean network
- POS
  - Useful in SAT solving and Boolean reasoning
  - Rarely used in circuit synthesis (due to the asymmetric characteristics of NMOS and PMOS)
- ROBDD
  - Canonical
  - Useful in Boolean reasoning
- Boolean network
  - Useful in multi-level logic optimization
- AIG
  - Useful in multi-level logic optimization and Boolean reasoning

Circuit to CNF Conversion

- Naive conversion of circuit to CNF:
  - Multiply out expressions of circuit until two level structure
  - Example: $y = x_1 \oplus x_2 \oplus x_2 \oplus \ldots \oplus x_n$ (Parity function)
    - circuit size is linear in the number of variables

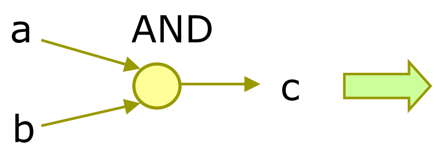


    - generated chess-board Karnaugh map
    - CNF (or DNF) formula has 2^{n-1} terms (exponential in #vars)

- Better approach:
  - Introduce one variable per circuit vertex
  - Formulate the circuit as a conjunction of constraints imposed on the vertex values by the gates
  - Uses more variables but size of formula is linear in the size of the circuit

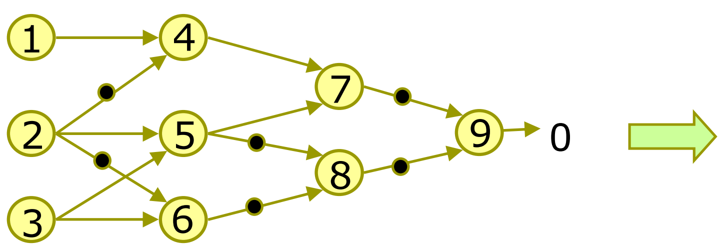
Circuit to CNF Conversion

- Example
  - Single gate:



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

  - Circuit of connected gates:



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

Is output always 0 ?

Justify to "1"

Circuit to CNF Conversion

- Circuit to CNF conversion
  - can be done in linear size (with respect to the circuit size) if intermediate variables can be introduced
  - may grow exponentially in size if no intermediate variables are allowed

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

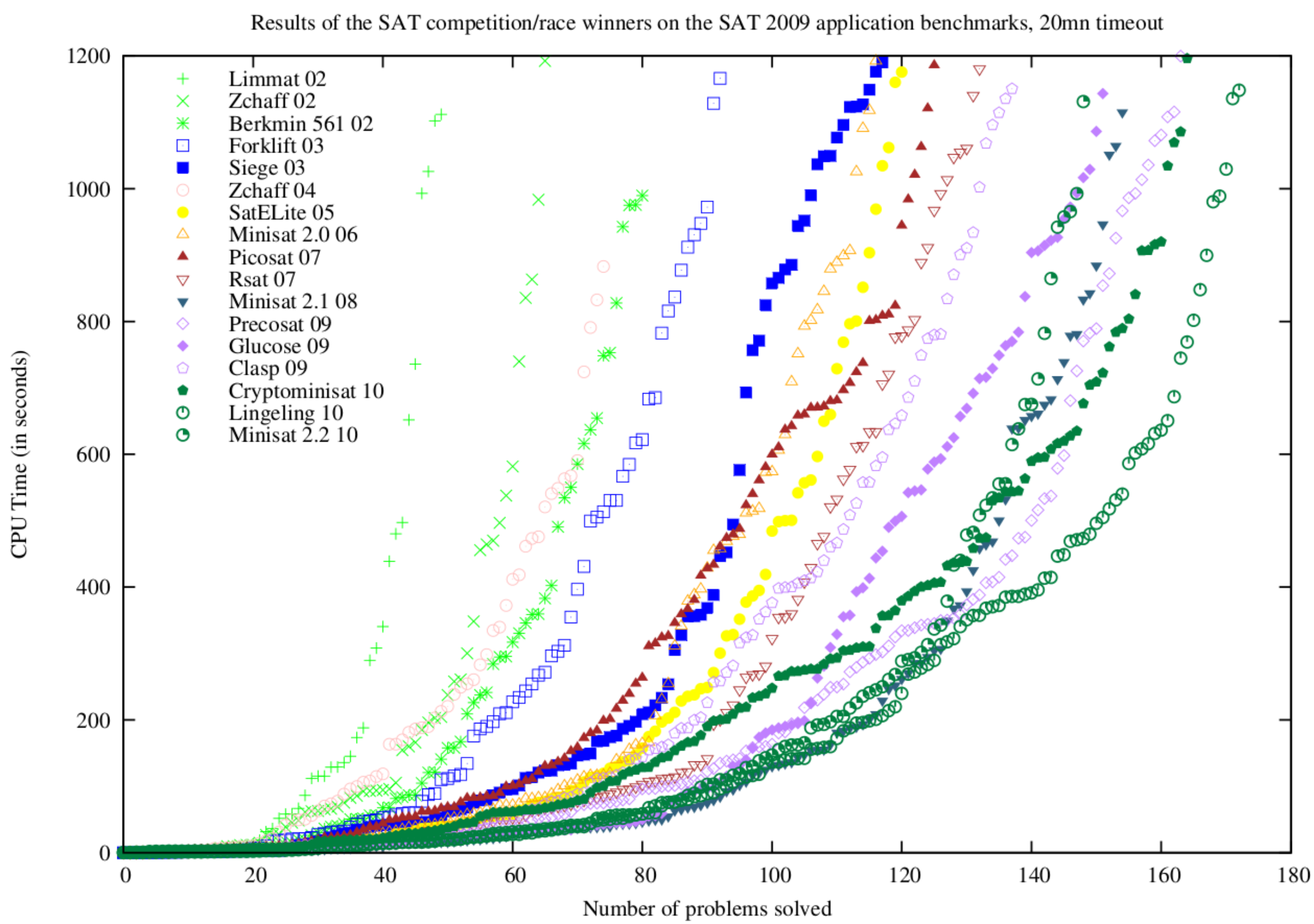
Satisfiability

- 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)]

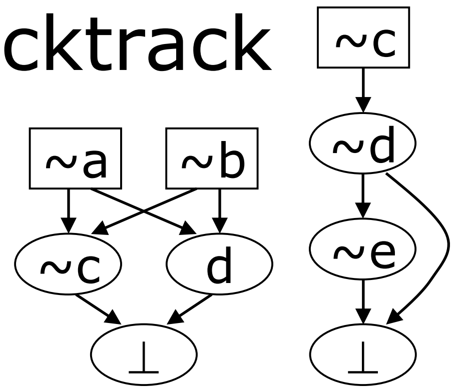
Pre-Modern SAT Procedure

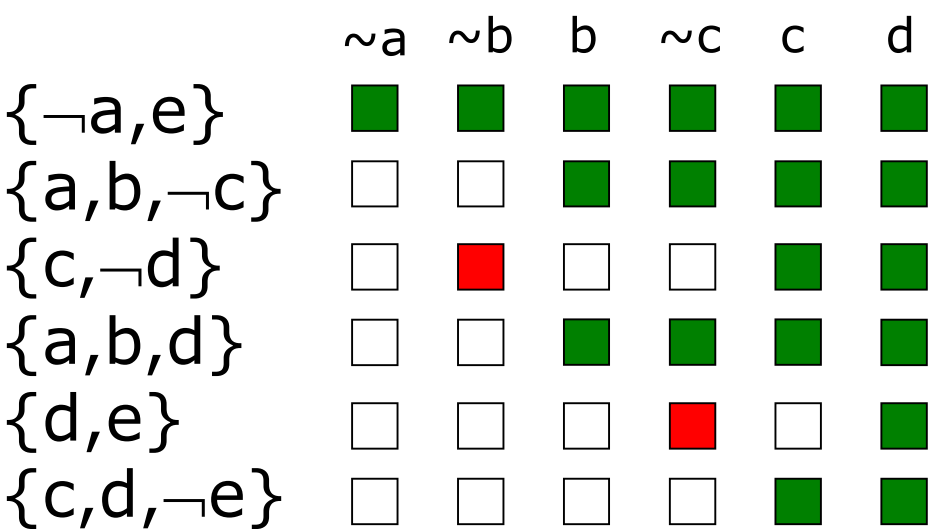
Algorithm `DPLL(Φ)`

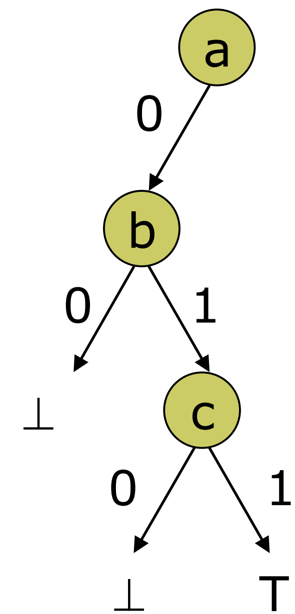
```
{
   while there is a unit clause {l} in Φ
      Φ = BCP(Φ, l);
   while there is a pure literal l in Φ
      Φ = assign(Φ, l);
   if all clauses of Φ satisfied   return true;
   if Φ has a conflicting clause   return false;
   l := choose_literal(Φ);
   return DPLL(assign(Φ,¬l)) ∨ DPLL(assign(Φ,l));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

DPLL Procedure

- Chorological backtrack
- E.g.







Modern SAT Procedure

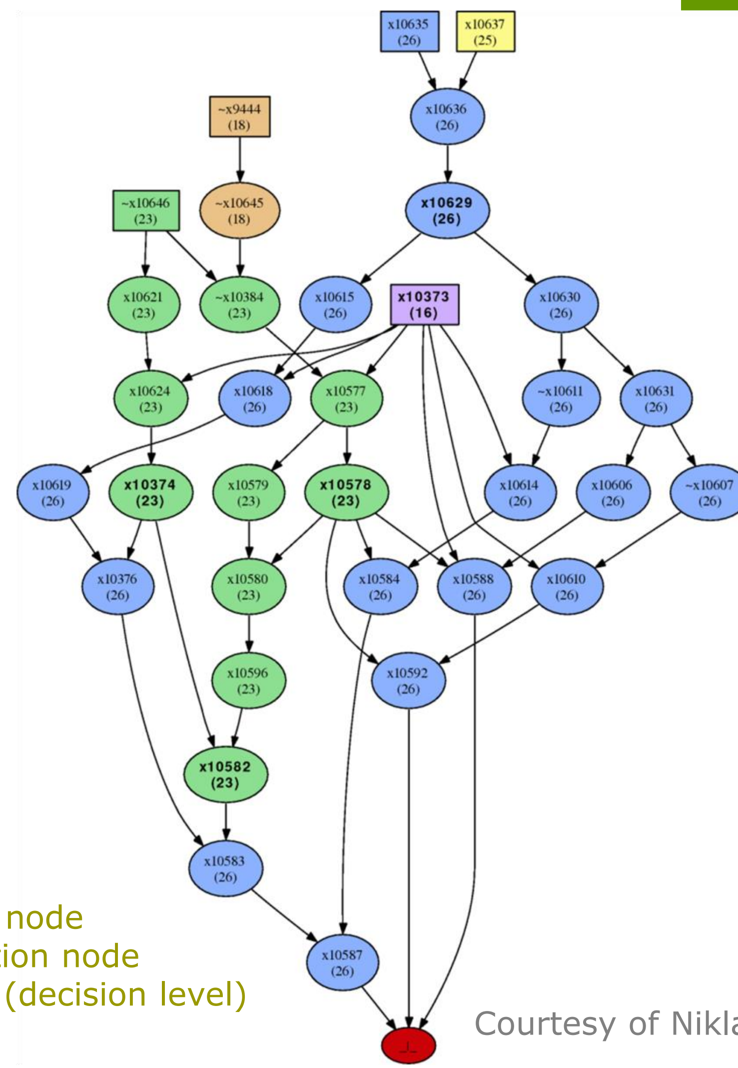
Algorithm `CDCL(Φ)`

```
{
  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.,
  $\{\neg x10587, \neg x10588, \neg x10592\}$
  …
  $\{\neg x10374, \neg x10582, \neg x10578, \neg x10373, \neg x10629\}$
  …
  $\{x10646, x9444, \neg x10373, \neg x10635, \neg x10637\}$



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 $C_2 \vee \neg x$:

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

  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

$$(a \lor b \lor c)(\neg a \lor c)(\neg b \lor \neg d)(\neg c)(c \lor d)$$

$$(b \lor c) \qquad (d)$$

$$(c \lor \neg d)$$

$$(\neg d)$$

$$()$$

SAT Certification

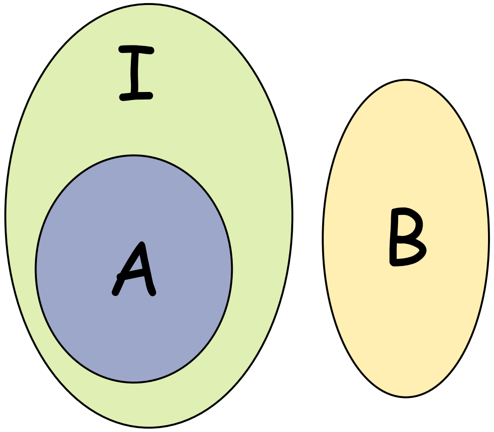
- True CNF
  - Satisfying assignment (model)
    - Verifiable in linear time

- 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

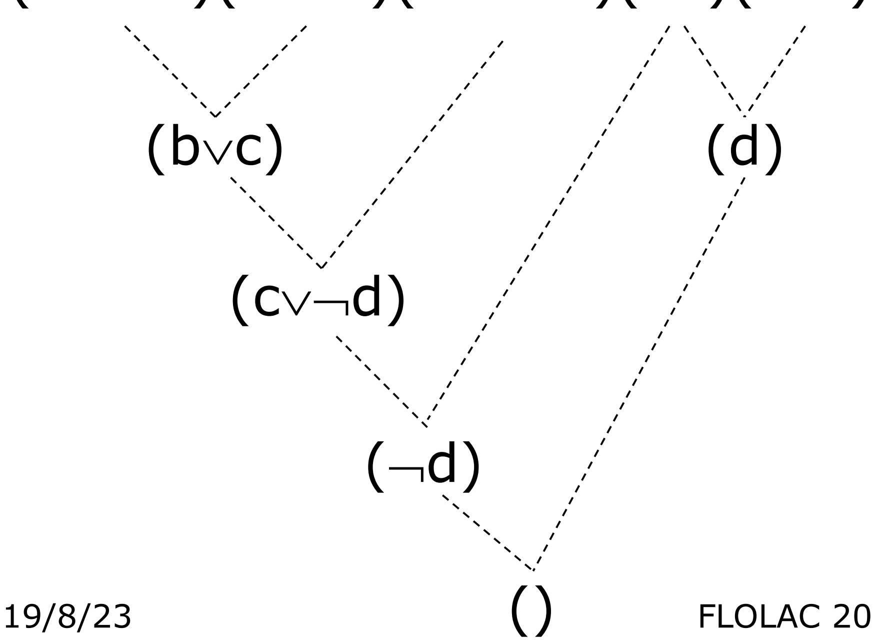


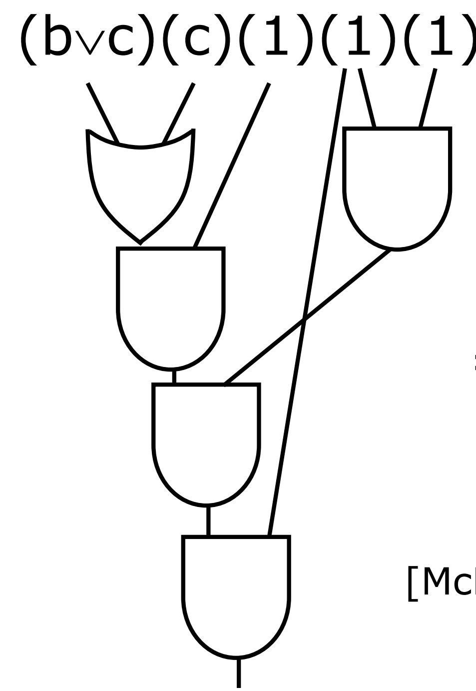
I is an abstraction of A

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

$$\overbrace{(a \vee b \vee c)(\neg a \vee c)}^{\varphi_A}\overbrace{(\neg b \vee \neg d)(\neg c)(c \vee d)}^{\varphi_B}$$





$= (b \vee c)$

[McMillan, 2003]

Incremental SAT Solving

- To solve, in a row, multiple CNF formulae, which are similar except for a few clauses, can we reuse the learnt clauses?
  - What if adding a clause to φ?
  - What if deleting a clause from φ?

Incremental SAT Solving

- MiniSat API
  - void *addClause*(Vec<Lit> clause)
  - bool *solve*(Vec<Lit> assumps)
  - bool *readModel*(Var x) – *for SAT results*
  - bool *assumpUsed*(Lit p) – *for UNSAT results*

  - The method *solve()* treats the literals in assumps as unit clauses to be temporary assumed during the SAT-solving.
  - More clauses can be added after *solve()* returns, then incrementally another SAT-solving executed.

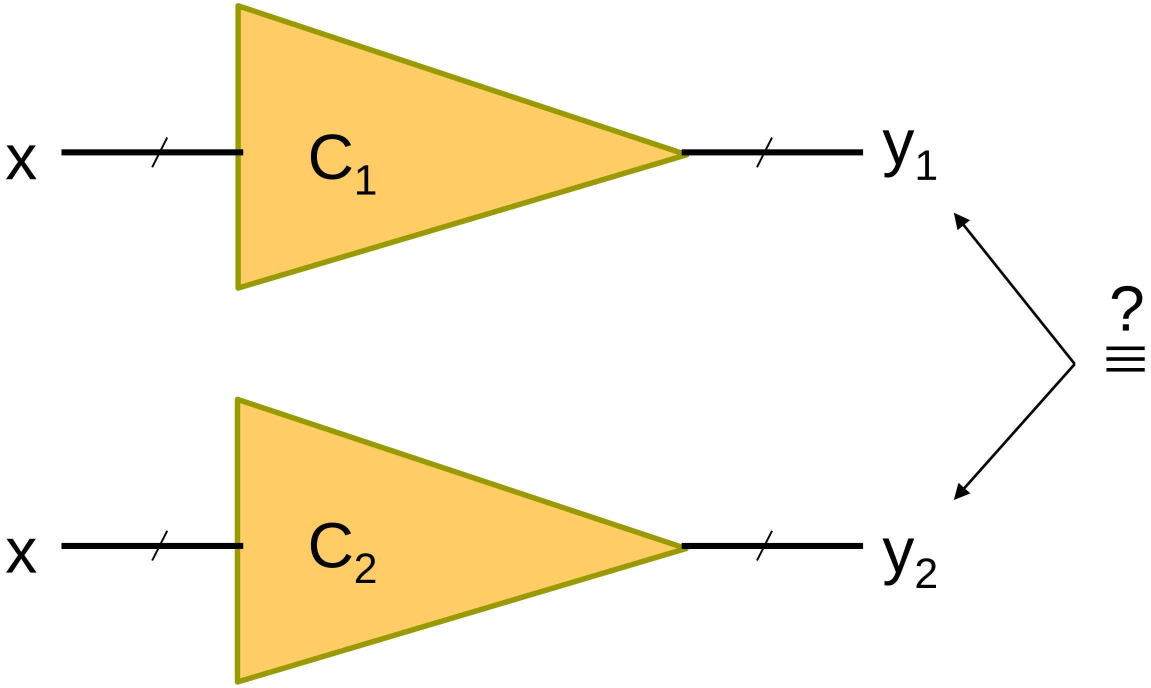
Courtesy of Niklas Een

SAT & Logic Synthesis

Equivalence Checking

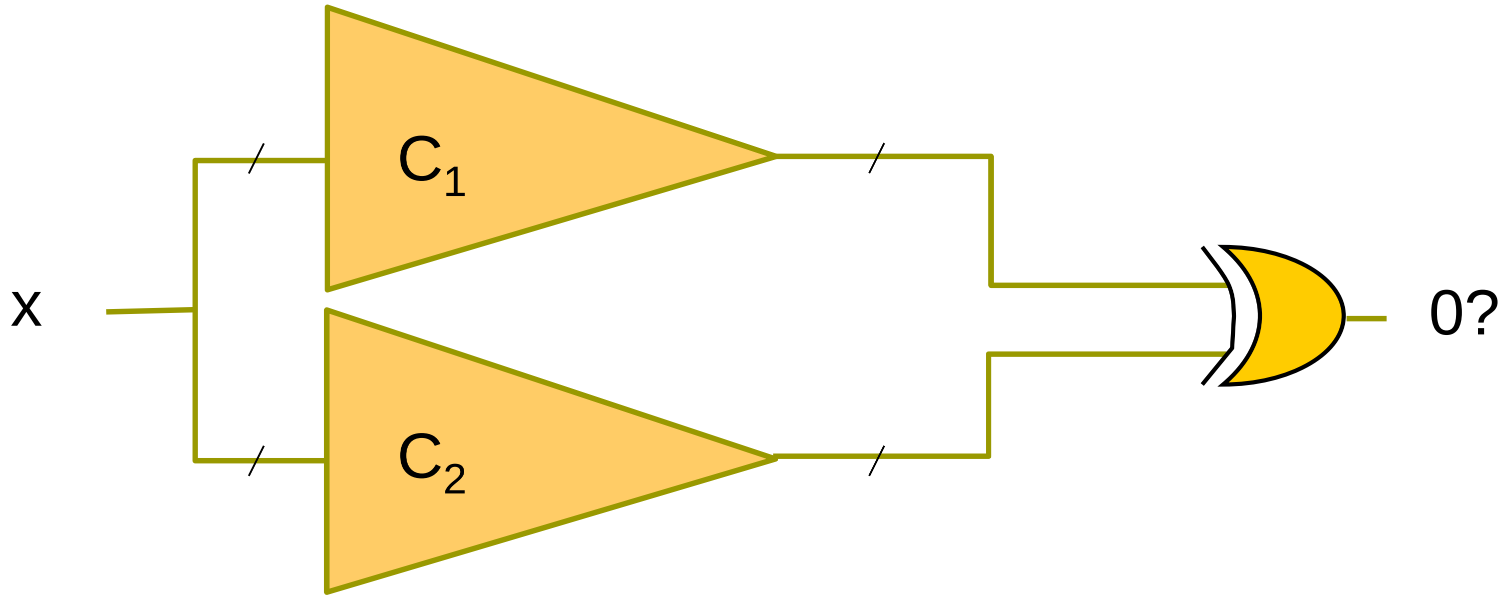
Combinational EC

- Given two combinational circuits C_1 and C_2, are their outputs equivalent under all possible input assignments?



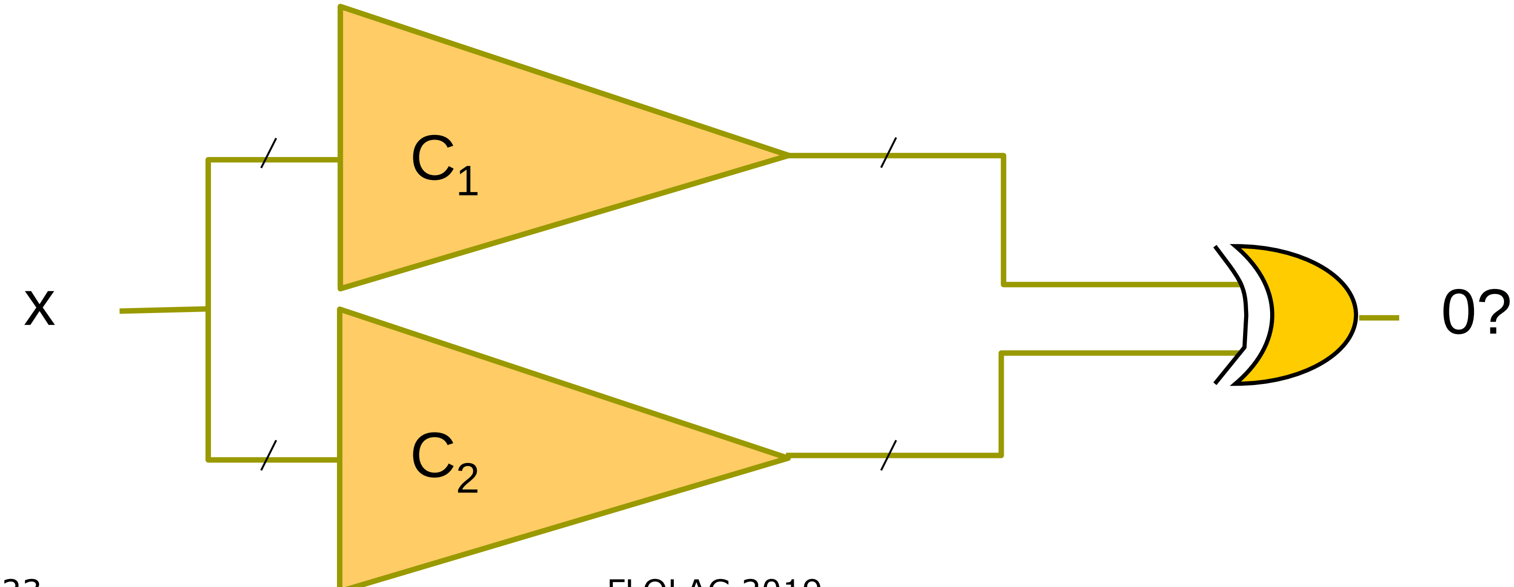
Miter for Combinational EC

- Two combinational circuits C_1 and C_2 are equivalent if and only if the output of their "miter" structure always produces constant 0



Approaches to Combinational EC

- Basic methods:
  - random simulation
    - good at identifying inequivalent signals
  - BDD-based methods
  - structural SAT-based methods



SAT & Logic Synthesis
Functional Dependency

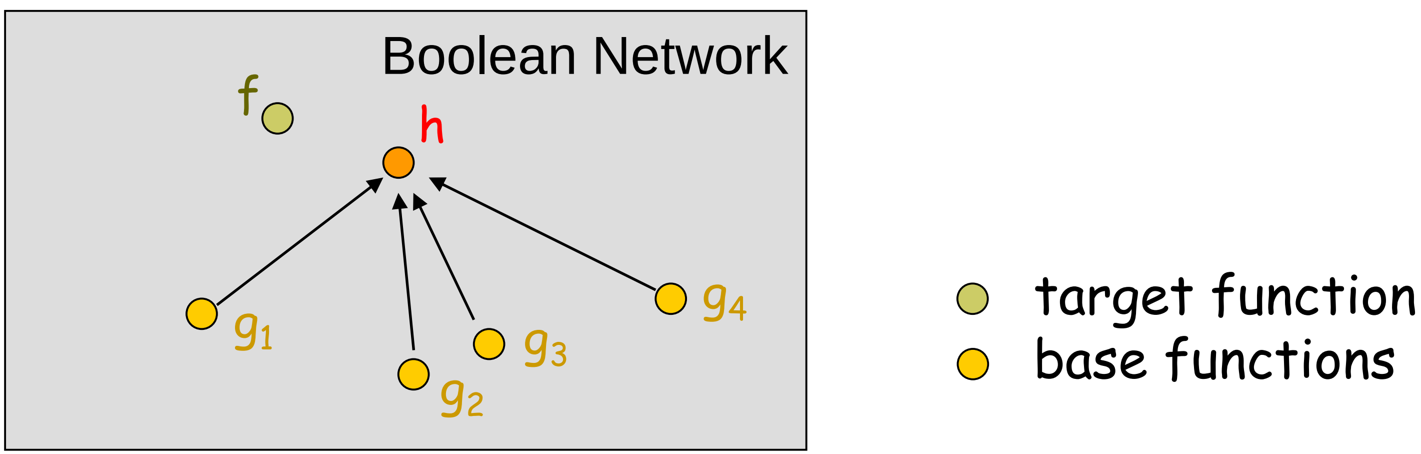
Functional Dependency

- $f(x)$ **functionally depends** on $g_1(x)$, $g_2(x)$, …, $g_m(x)$ if $f(x) = h(g_1(x), g_2(x), …, g_m(x))$, denoted $h(G(x))$
  - Under what condition can function f be expressed as some function h over a set $G=\{g_1,…,g_m\}$ of functions ?
  - 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

Motivation

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

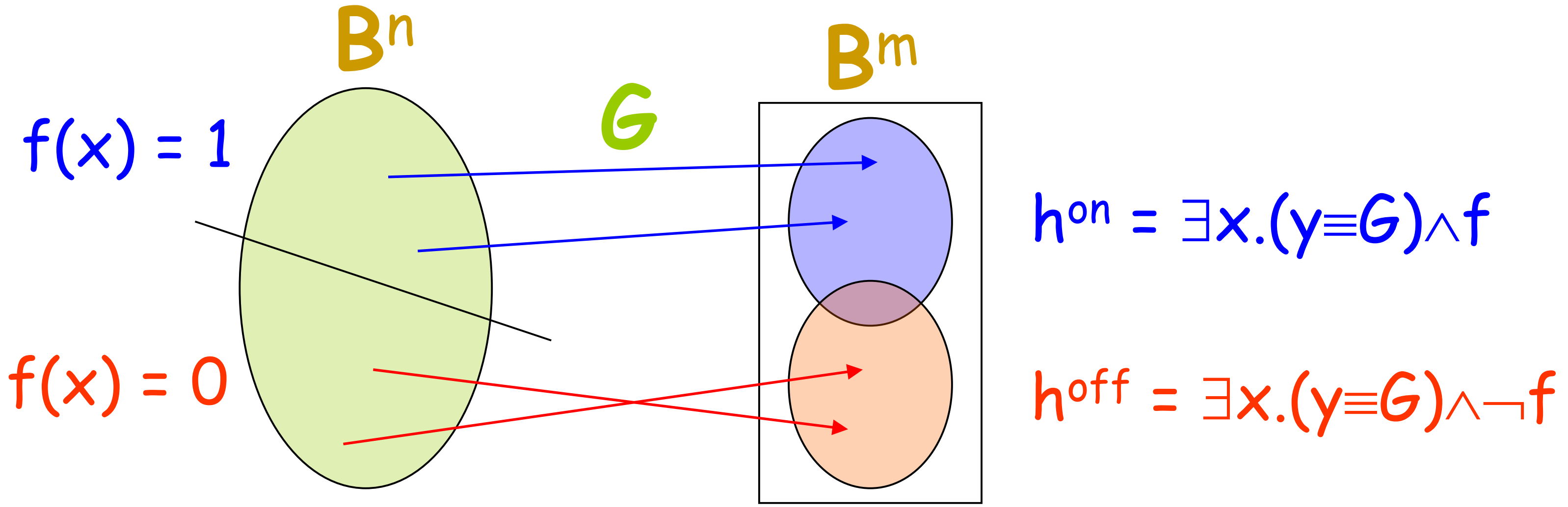


BDD-Based Computation

- BDD-based computation of h

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

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



BDD-Based Computation

- Pros
  - Exact computation of h^{on} and h^{off}
  - Better support for don't care minimization
- Cons
  - 2 image computations for every choice of G
  - Inefficient when $|G|$ is large or when there are many choices of G

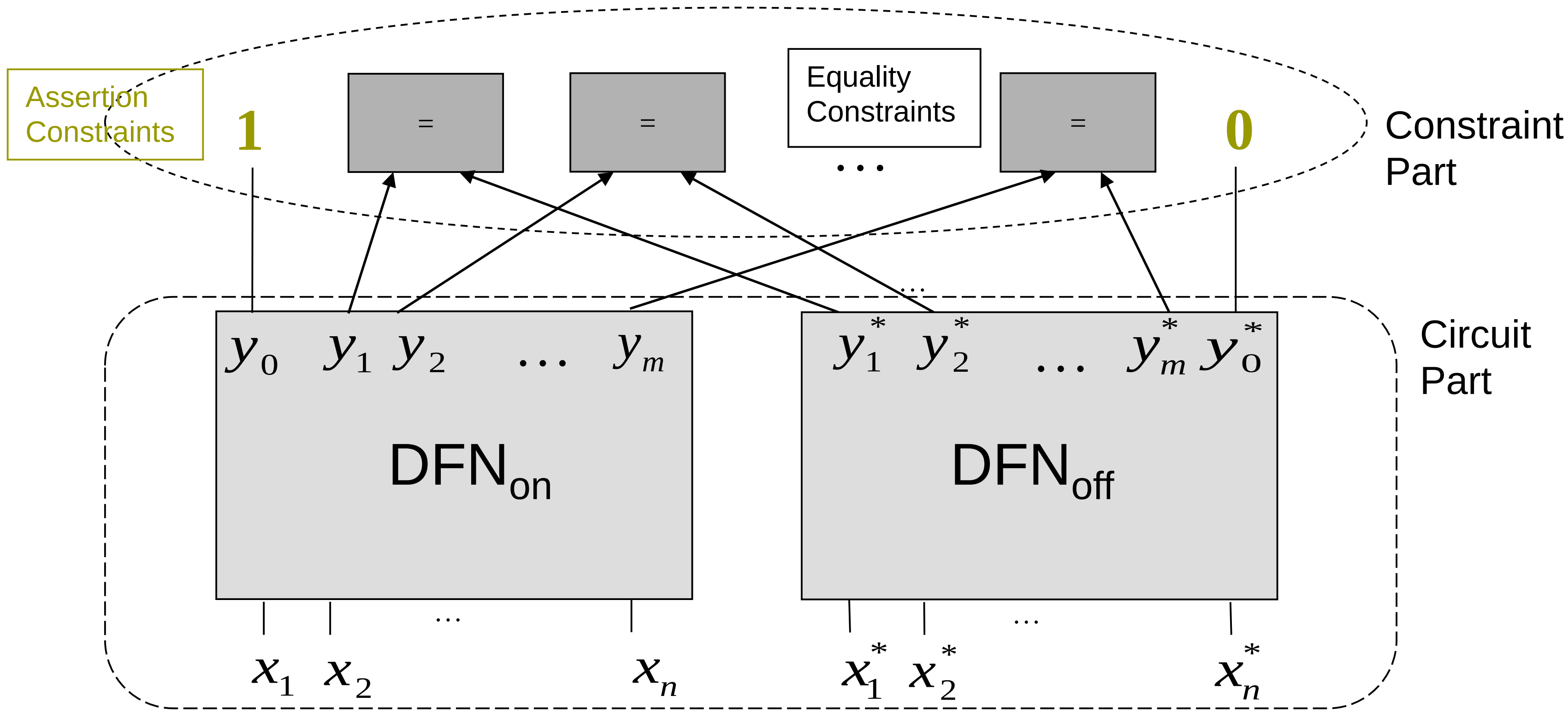
SAT-Based Computation

- h exists $\Leftrightarrow$
  $\nexists a,b$ such that $f(a)\neq f(b)$ and $G(a)=G(b)$,
  i.e., $(f(x)\neq f(x^*))\wedge(G(x)\equiv G(x^*))$ is UNSAT

- How to derive h? How to select G?

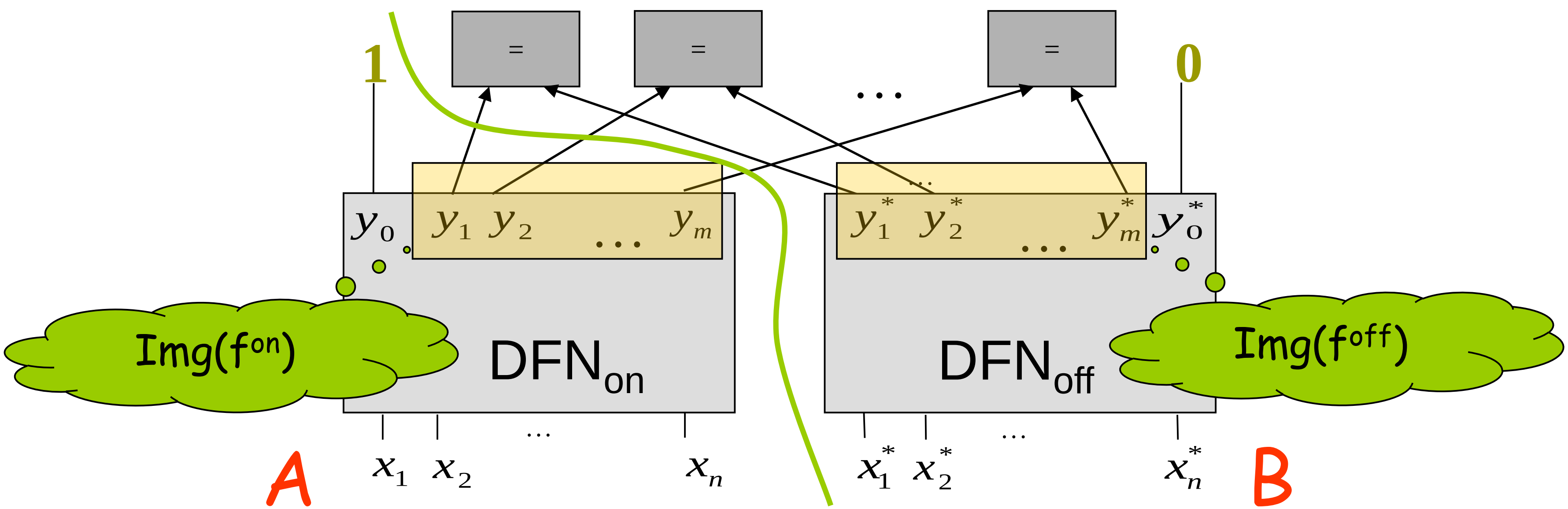
SAT-Based Computation

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



Deriving *h* with Craig Interpolation

- 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



Incremental SAT Solving

- Controlled equality constraints

  $(y_i \equiv y_i^*) \rightarrow (\neg y_i \vee y_i^* \vee \alpha_i)(y_i \vee \neg y_i^* \vee \alpha_i)$

  with auxiliary variables α_i

  α_i = true $\Rightarrow$ i^{th} equality constraint is disabled

  - Fast switch between target and base functions by unit assumptions over control variables
  - Fast enumeration of different base functions
  - Share learned clauses

SAT vs. BDD

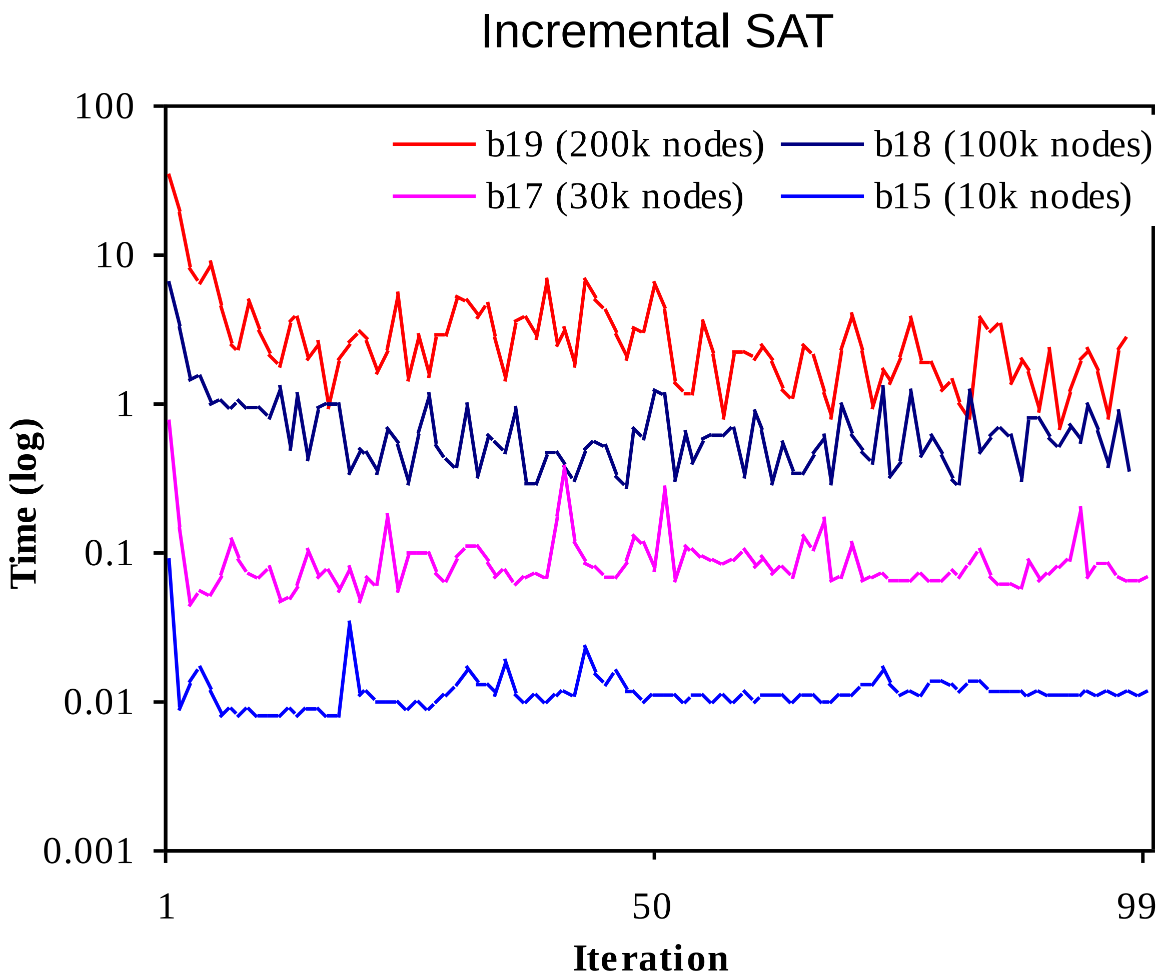
- SAT
  - Pros
    - Detect multiple choices of G automatically
    - Scalable to large $|G|$
    - Fast enumeration of different target functions f
    - Fast enumeration of different base functions G
  - Cons
    - Single feasible implementation of h
- BDD
  - Cons
    - Detect one choice of G at a time
    - Limited to small $|G|$
    - Slow enumeration of different target functions f
    - Slow enumeration of different base functions G
  - Pros
    - All possible implementations of h

Practical Evaluation

SAT vs. BDD

		Original			Retimed			SAT (original)		BDD (original)		SAT (retimed)		BDD (retimed)	
Circuit	#Nodes	#FF.	#Dep-S	#Dep-B	#FF.	#Dep-S	#Dep-B	Time	Mem	Time	Mem	Time	Mem	Time	Mem
s5378	2794	179	52	25	398	283	173	1.2	18	1.6	20	0.6	18	7	51
s9234.1	5597	211	46	x	459	301	201	4.1	19	x	x	1.7	19	194.6	149
s13207.1	8022	638	190	136	1930	802	x	15.6	22	31.4	78	15.3	22	x	x
s15850.1	9785	534	18	9	907	402	x	23.3	22	82.6	94	7.9	22	x	x
s35932	16065	1728	0	--	2026	1170	--	176.7	27	1117	164	78.1	27	--	--
s38417	22397	1636	95	--	5016	243	--	270.3	30	--	--	123.1	32	--	--
s38584	19407	1452	24	--	4350	2569	--	166.5	21	--	--	99.4	30	1117	164
b12	946	121	4	2	170	66	33	0.15	17	12.8	38	0.13	17	2.5	42
b14	9847	245	2	--	245	2	--	3.3	22	--	--	5.2	22	--	--
b15	8367	449	0	--	1134	793	--	5.8	22	--	--	5.8	22	--	--
b17	30777	1415	0	--	3967	2350	--	119.1	28	--	--	161.7	42	--	--
b18	111241	3320	5	--	9254	5723	--	1414	100	--	--	2842.6	100	--	--
b19	224624	6642	0	--	7164	337	--	8184.8	217	--	--	11040.6	234	--	--
b20	19682	490	4	--	1604	1167	--	25.7	28	--	--	36	30	--	--
b21	20027	490	4	--	1950	1434	--	24.6	29	--	--	36.3	31	--	--
b22	29162	735	6	--	3013	2217	--	73.4	36	--	--	90.6	37	--	--

Practical Evaluation



Quantified Boolean 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

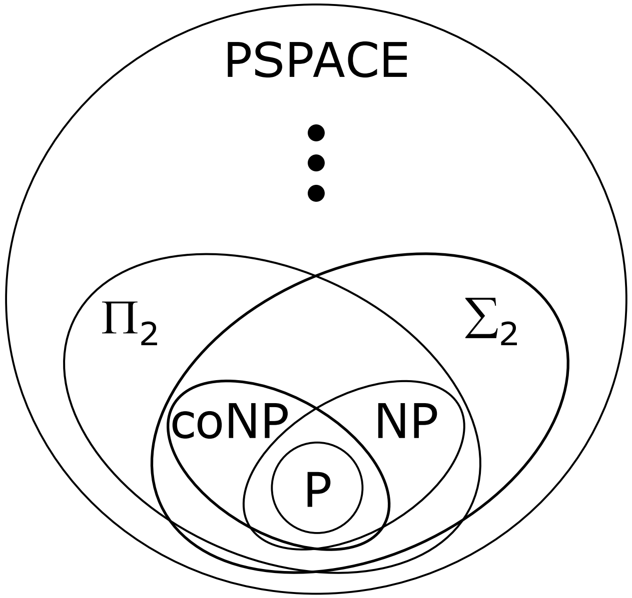
- Quantification order matters in a QBF
- A variable x_i in $(Q_1\, x_1, \ldots, Q_i\, x_i, \ldots, Q_n\, x_n.\ \varphi)$ is of **level** k if there are k quantifier alternations (i.e., changing from $\forall$ to $\exists$ or from $\exists$ to $\forall$) from Q_1 to Q_i.
  - Example

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

    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

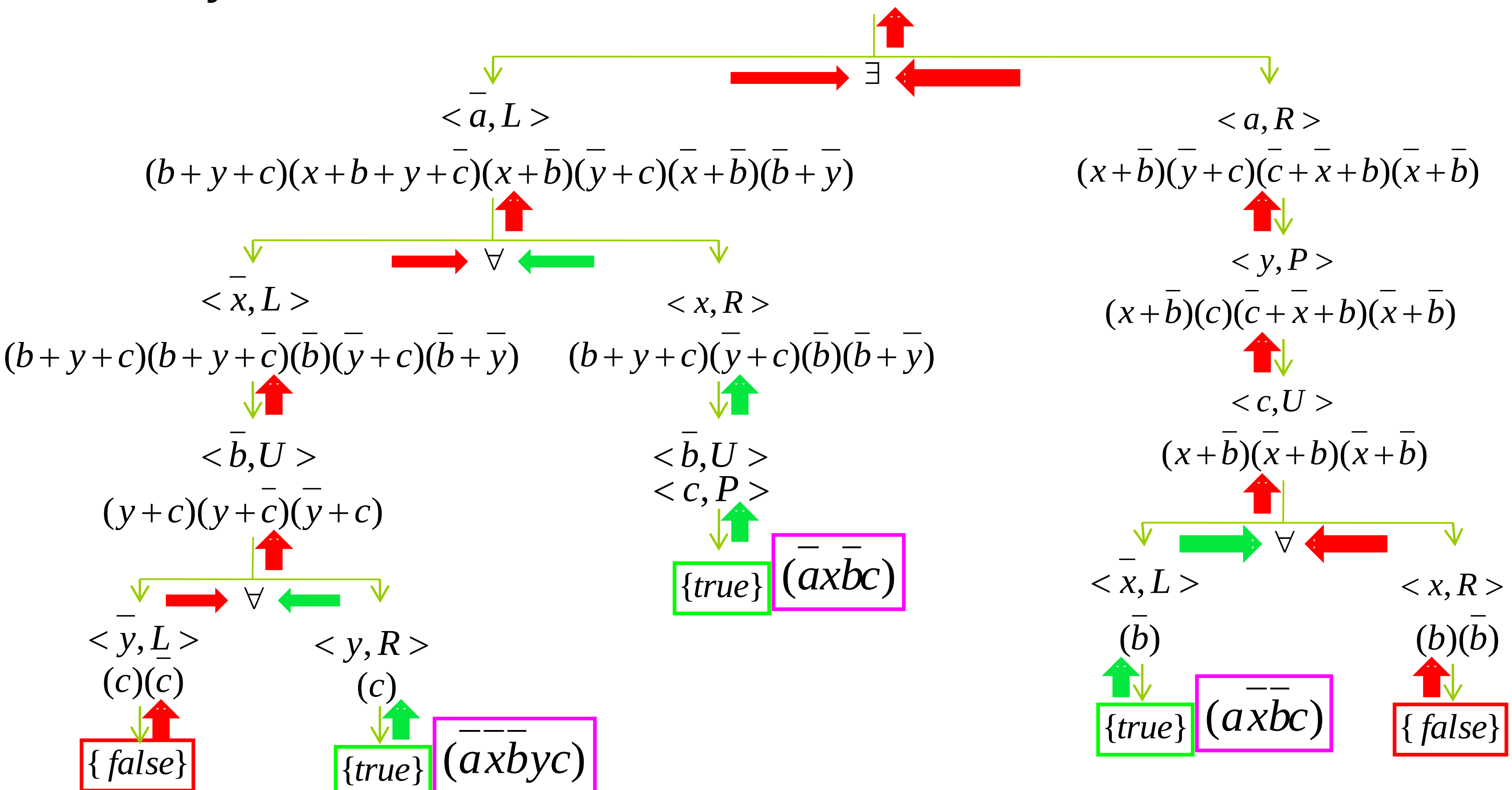
- QBF solver choices
  - Data structures for formula representation
    - **Prenex** vs. non-prenex
    - **Normal form** vs. non-normal form
      - CNF, NNF, BDD, AIG, etc.
  - Solving mechanisms
    - **Search**, Q-resolution, Skolemization, quantifier elimination, etc.
  - Preprocessing techniques

- 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
      - Example
        $\forall a\ \exists b\ \exists c\ \forall d\ \exists e.\ (a+c)(\neg a+\neg c)(b+\neg c+e)(\neg b)(c+d+\neg e)(\neg c+e)(\neg d+e)$
        from 00101, we learn cube $\neg a \neg b c \neg d$ (can be further simplified to $\neg a$)

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**

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

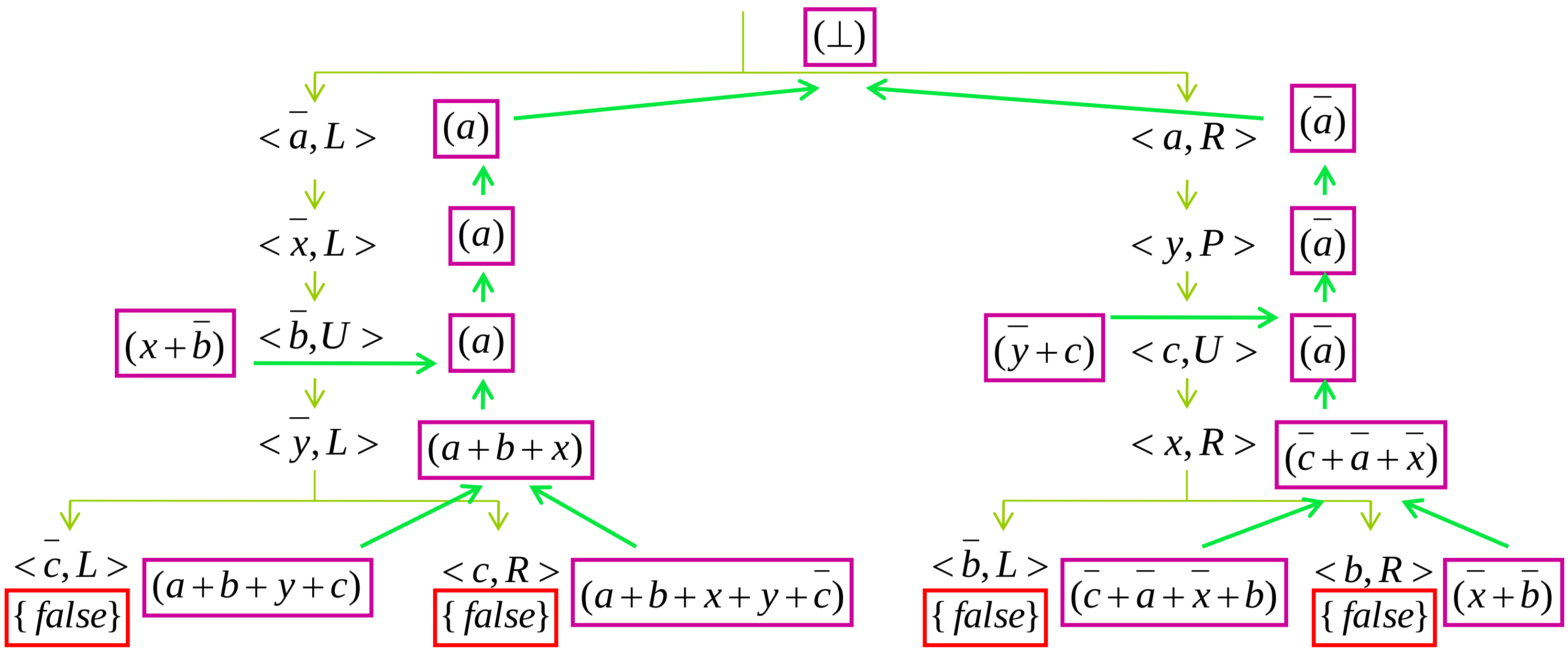
  where operator ∀-RED removes from $C_1 \vee C_2$ the universally (∀) quantified variables whose quantification levels are greater than any of the existentially (∃) quantified variables in $C_1 \vee C_2$
  - E.g.,
    prefix: $\forall a\ \exists b\ \forall c\ \forall d\ \exists e$
    $\forall$-RED$(a+b+c+d)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

- 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

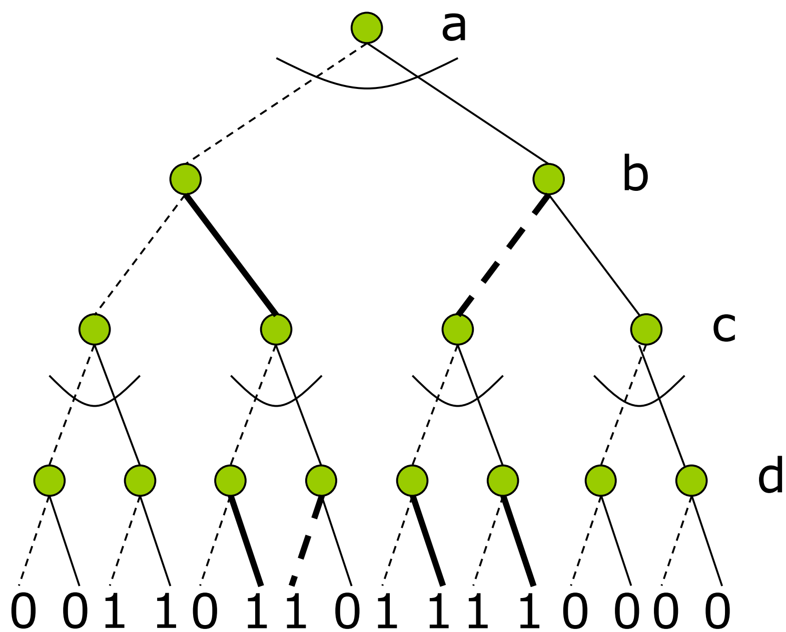
- Skolemization and Skolem normal form
  - Existentially quantified variables are replaced with function symbols
  - QBF prefix contains only two quantification levels
    - $\exists$ function symbols, $\forall$ variables

- 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
  - Skolemization-based solvers (e.g., sKizzo, squolem, Ebddres) can provide SFs
  - Search-based solvers (e.g., DepQBF) can be instrumented to provide resolution proofs

QBF Certification

- Solvers and certificates (prior to 2011)

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 (prior to 2011)

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

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

QBF Certification

- Unified QBF certification

True QBF $\xleftrightarrow{\text{formula negation}}$ **False QBF**

Cube resolution proof $\xrightarrow{\text{ResQu}}$ Skolem function (model)

Clause resolution proof $\xrightarrow{\text{ResQu}}$ Herbrand function (countermodel)

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

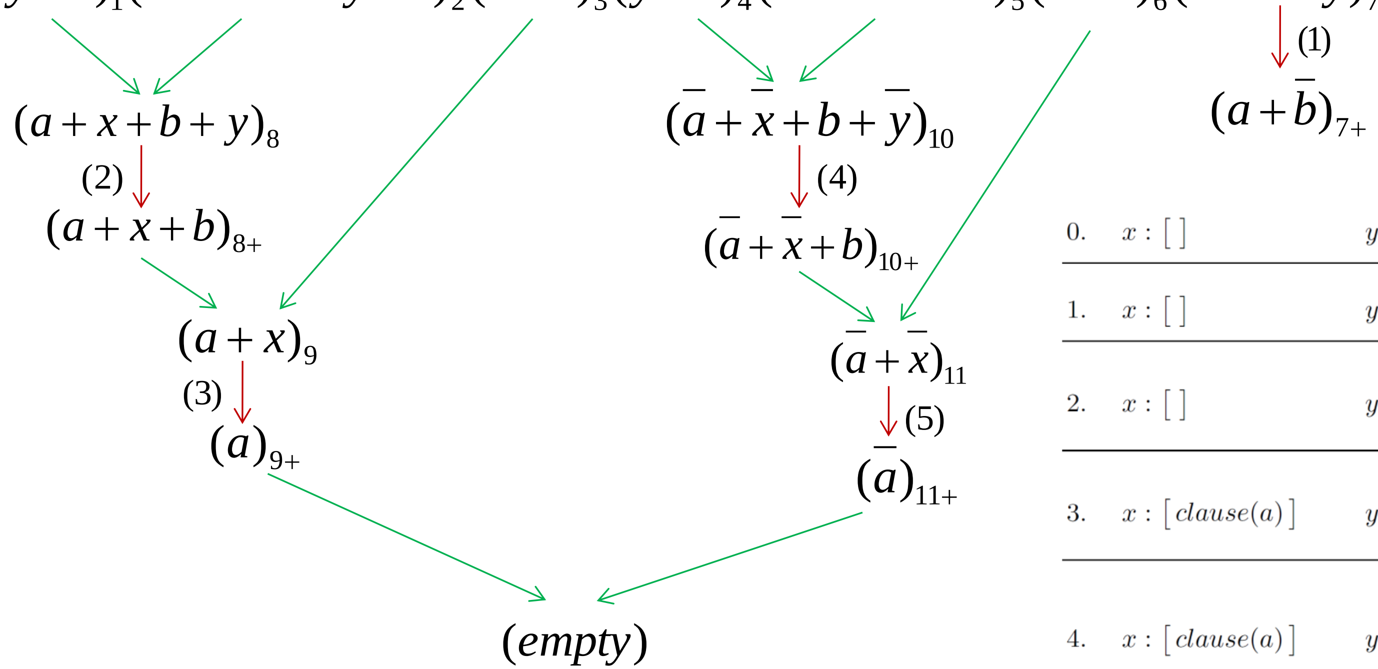
Countermodel_construct

```
input: a false QBF \Phi and its clause-resolution DAG G_\Pi(V_\Pi, E_\Pi)
output: a countermodel in RFAO formulas
begin
  foreach universal variable x of \Phi
    RFAO_node_array[x] := \emptyset;
  foreach vertex v of G_\Pi 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 \Phi
        simplify RFAO_node_array[x];
      return RFAO_node_array's;
end
```

ResQu

- Example
  - ∃a∀x∃b∀y∃c

$$(a+b+y+c)_1(a+x+b+y+\bar{c})_2(x+\bar{b})_3(\bar{y}+c)_4(\bar{a}+\bar{x}+b+\bar{c})_5(\bar{x}+\bar{b})_6(a+\bar{b}+\bar{y})_7$$



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

QBF Certification

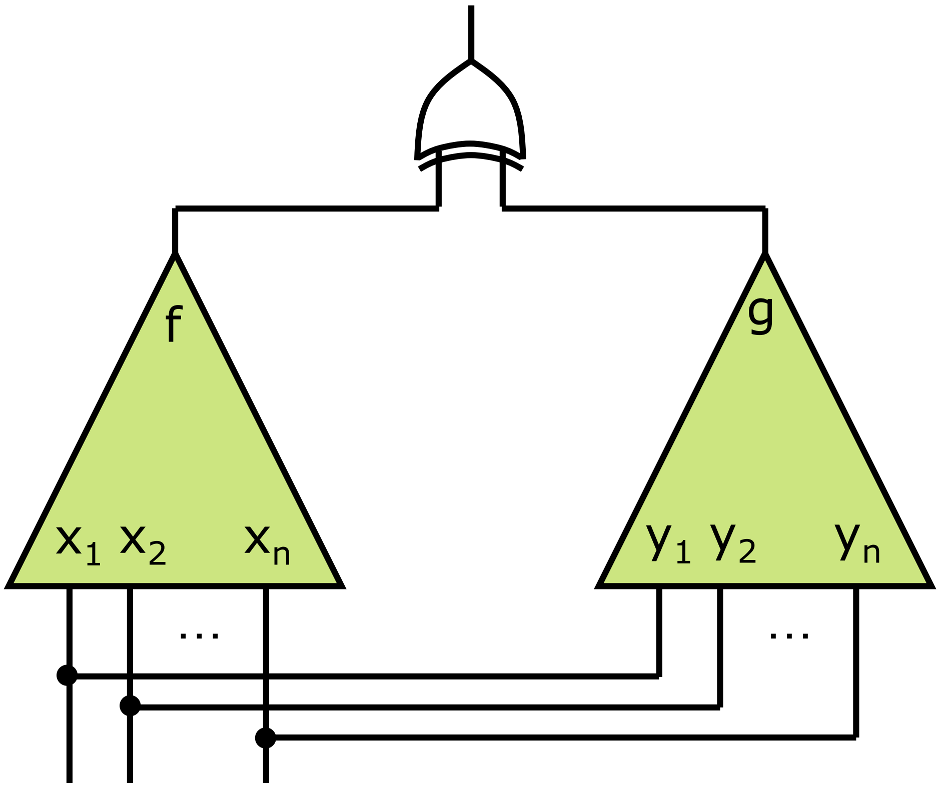
- Applications of Skolem/Herbrand functions
  - Program synthesis
  - Winning strategy synthesis in two player games
  - Plan derivation in AI
  - Logic synthesis
  - …

QSAT & Logic Synthesis

Boolean Matching

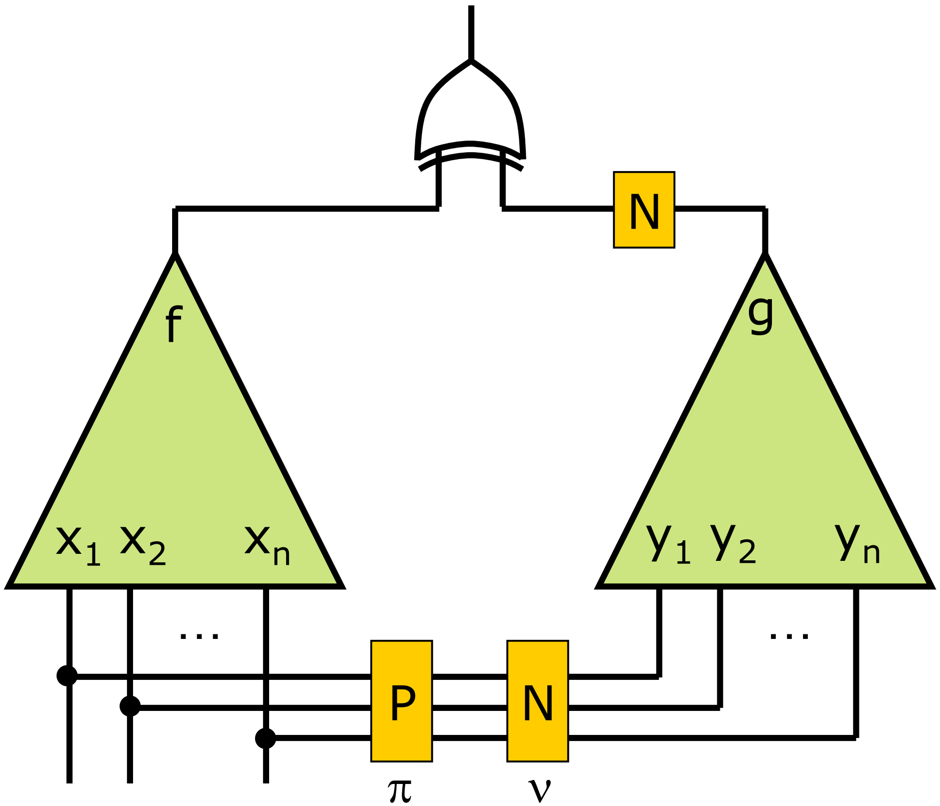
Introduction

- Combinational equivalence checking (CEC)
  - Known input correspondence
  - coNP-complete
  - Well solved in practical applications



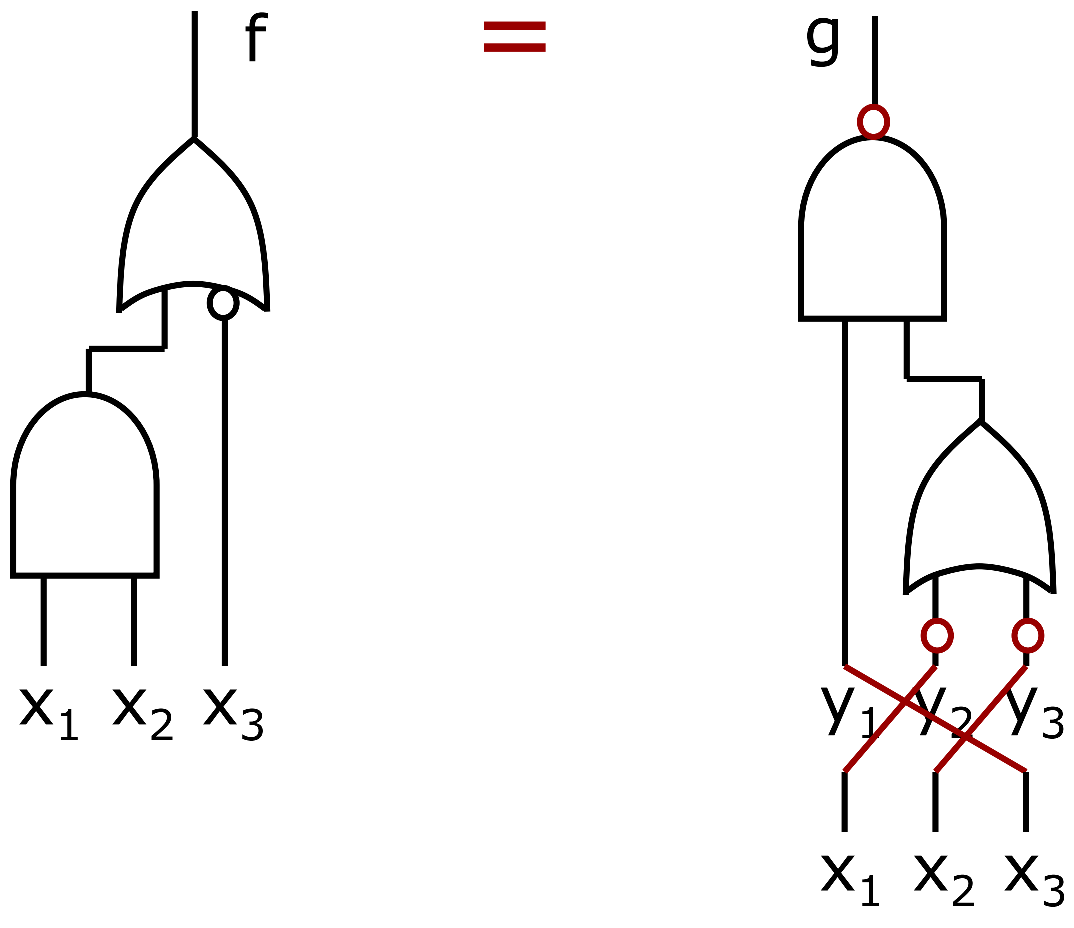
Introduction

- Boolean matching
  - P-equivalence
    - Unknown input permutation
    - O(n!) CEC iterations
  - NP-equivalence
    - Unknown input negation and permutation
    - O(2^nn!) CEC iterations
  - NPN-equivalence
    - Unknown input negation, input permutation, and output negation
    - O(2^{n+1}n!) CEC iterations



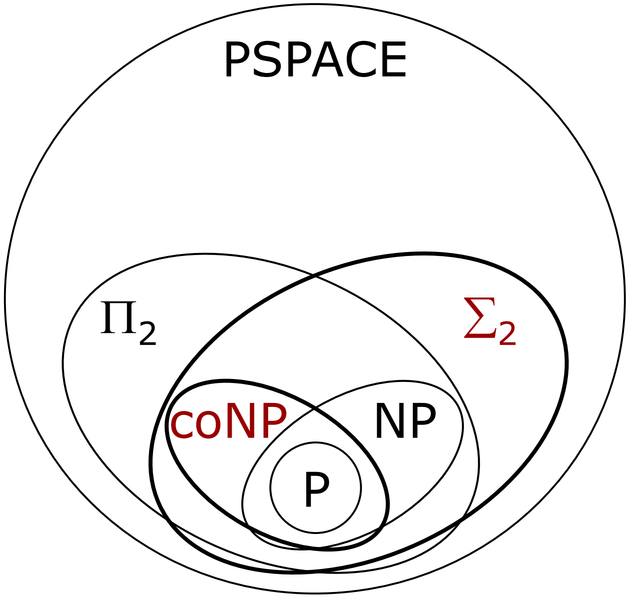
Introduction

- Example



Introduction

- Motivations
  - Theoretically
    - Complexity in between
      coNP (for all …) and
      Σ_2 (there exists … for all …)
      in the Polynomial Hierarchy (PH)
      - Special candidate to test PH collapse
    - Known as Boolean congruence/isomorphism dating back to the 19th century
  - Practically
    - Broad applications
      - Library binding
      - FPGA technology mapping
      - Detection of generalized symmetry
      - Logic verification
      - Design debugging/rectification
      - Functional engineering change order
    - Intensively studied over the last two decades



Introduction

- Prior methods

	Complete ?	Function type	Equivalence type	Solution type	Scalability
Spectral methods	yes	CS	mostly P	one	– –
Signature based methods	no	mostly CS	P/NP	N/A	– ~ ++
Canonical-form based methods	yes	CS	mostly P	one	+
SAT based methods	yes	CS	mostly P	one/all	+
BooM (QBF/SAT-like)	yes	CS / IS	NPN	one/all	++

CS: completely specified
IS: incompletely specified

BooM: A Fast Boolean Matcher

- Features of BooM
  - General computation framework
  - Effective search space reduction techniques
    - **Dynamic learning** and **abstraction**
  - Theoretical SAT-iteration upper-bound:

$$\cancel{O(2^n n!)} \qquad O(2^{2^n})$$

Formulation

- Reduce NPN-equiv to 2 NP-equiv checks
  - Matching f and g; matching f and $\neg$g
- 2^{nd} order formula of NP-equivalence

  $\exists \nu \circ \pi, \forall x\ ((f_c(x) \wedge g_c(\nu \circ \pi(x))) \Rightarrow (f(x) \equiv g(\nu \circ \pi(x))))$

  - f_c and g_c are the care conditions of f and g, respectively
- Need 1^{st} order formula instead for SAT solving

Formulation

- 0-1 matrix representation of $\nu \circ \pi$

$$\begin{array}{c c} & \begin{array}{cccccccc} x_1 & \neg x_1 & x_2 & \neg x_2 & \cdots & x_n & \neg x_n \end{array} \\ \begin{array}{c} y_1 \\ y_2 \\ \vdots \\ y_n \end{array} & \begin{pmatrix} a_{11} & b_{11} & a_{12} & b_{12} & \cdots & a_{1n} & b_{1n} \\ a_{21} & b_{21} & a_{22} & b_{22} & \cdots & a_{2n} & b_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & b_{n1} & a_{n2} & b_{n2} & \cdots & a_{nn} & b_{nn} \end{pmatrix} \end{array} \quad \sum = 1$$

$\sum = 1$

$a_{ij} \Rightarrow (x_j \equiv y_i)$

$b_{ij} \Rightarrow (\neg x_j \equiv y_i)$

Formulation

- Quantified Boolean formula (QBF) for NP-equivalence

$$\exists a, \exists b, \forall x, \forall y\ (\varphi_C \wedge \varphi_A \wedge ((f_c \wedge g_c) \Rightarrow (f \equiv g))$$

  - φ_C: cardinality constraint
  - φ_A: $\bigwedge_{i,j} (a_{ij} \Rightarrow (y_i \equiv x_j))\ (b_{ij} \Rightarrow (y_i \equiv \neg x_j))$

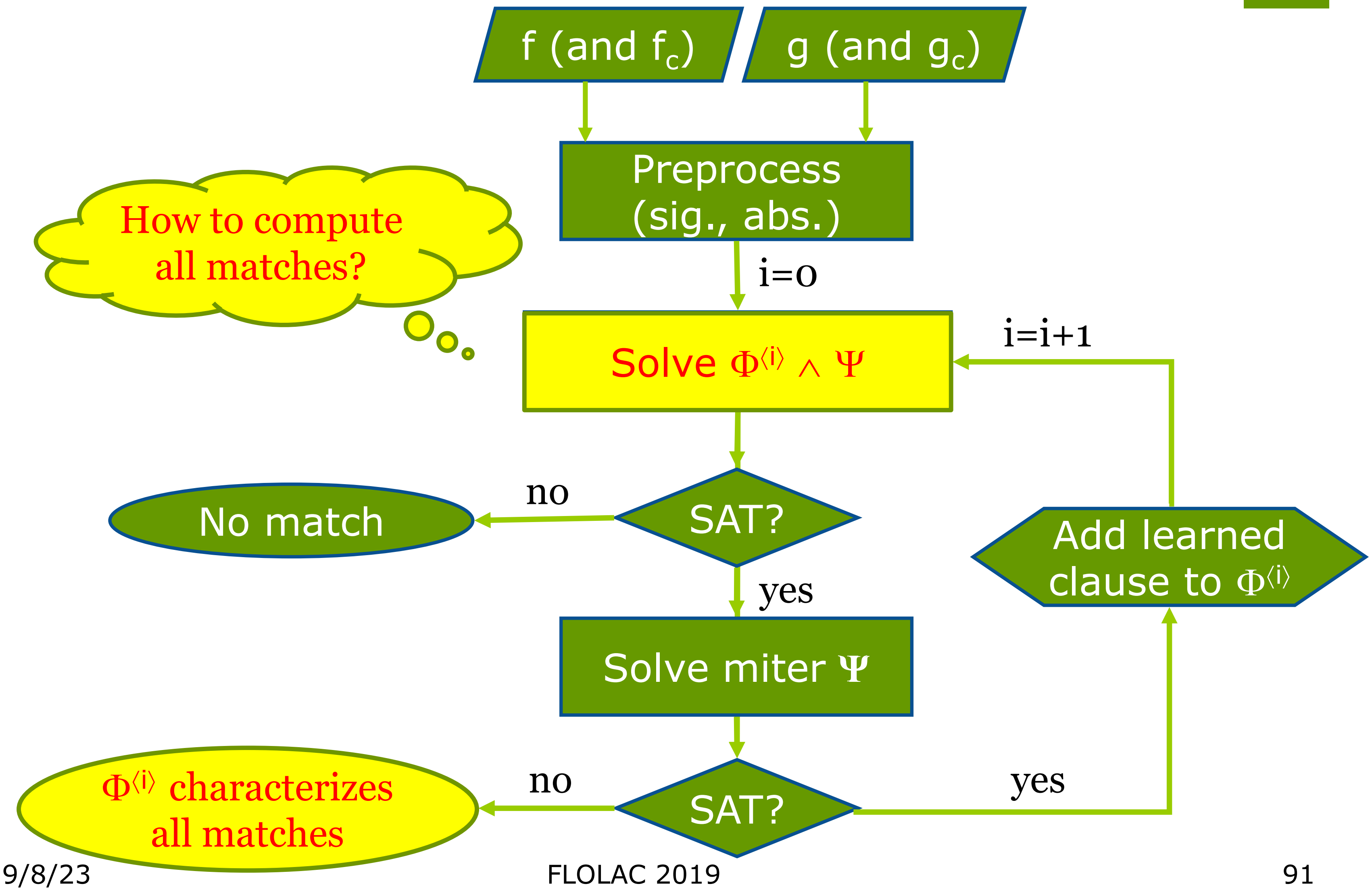
- Look for an assignment to a- and b-variables that satisfies φ_C and makes the **miter constraint**

$$\Psi = \varphi_A \wedge (f \neq g) \wedge f_c \wedge g_c$$

  unsatisfiable

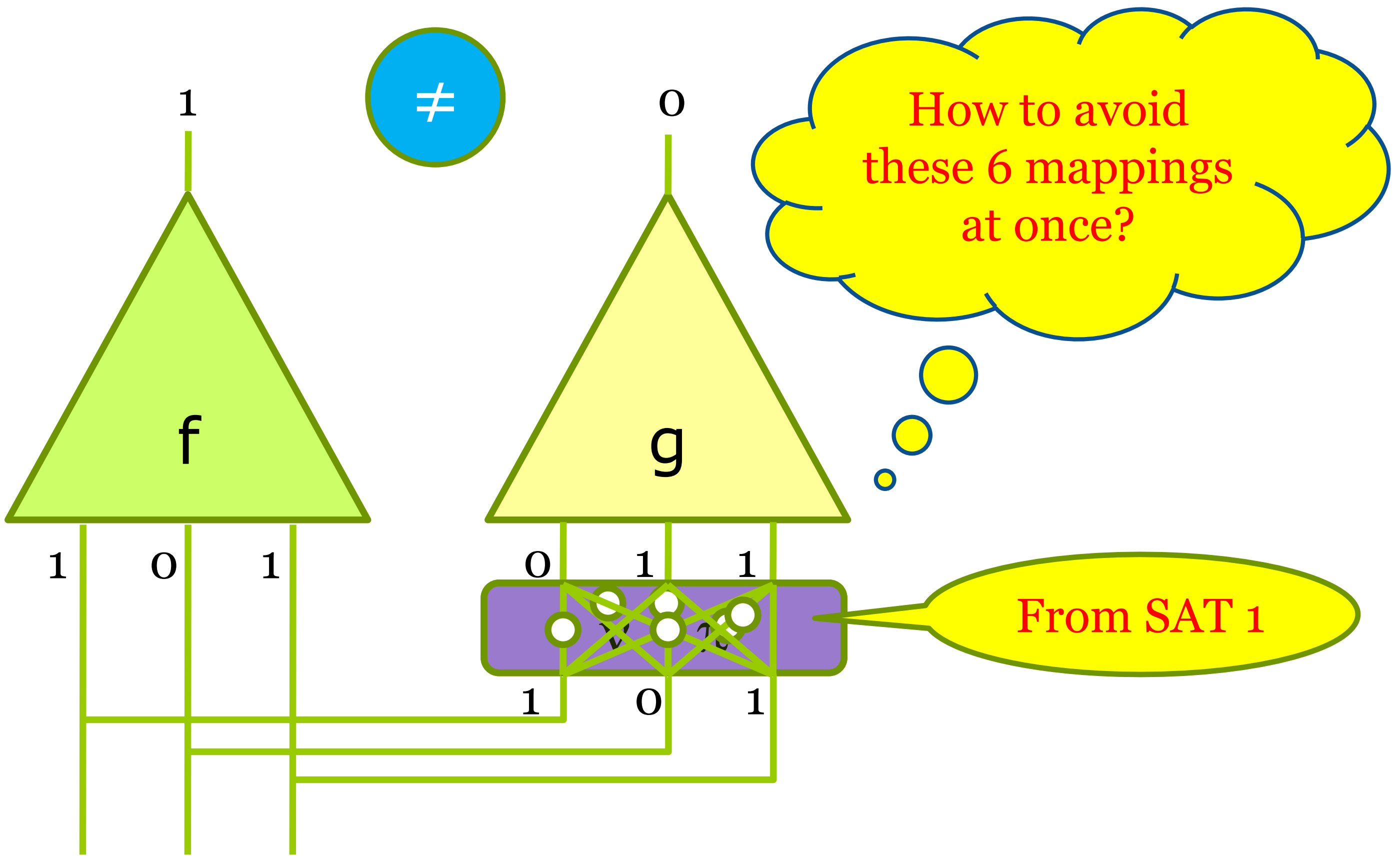
- Refine φ_C iteratively in a sequence $\Phi^{\langle 0 \rangle}, \Phi^{\langle 1 \rangle}, \ldots, \Phi^{\langle k \rangle}$, for $\Phi^{\langle i+1 \rangle} \Rightarrow \Phi^{\langle i \rangle}$ through **conflict-based learning**

BooM Flow


f (and f_c)
g (and g_c)
Preprocess (sig., abs.)
How to compute all matches?
i=0
Solve $\Phi^{\langle i \rangle} \wedge \Psi$
i=i+1
no
No match
SAT?
Add learned clause to $\Phi^{\langle i \rangle}$
yes
Solve miter Ψ
no
$\Phi^{\langle i \rangle}$ characterizes all matches
SAT?
yes

NP-Equivalence
Conflict-based Learning

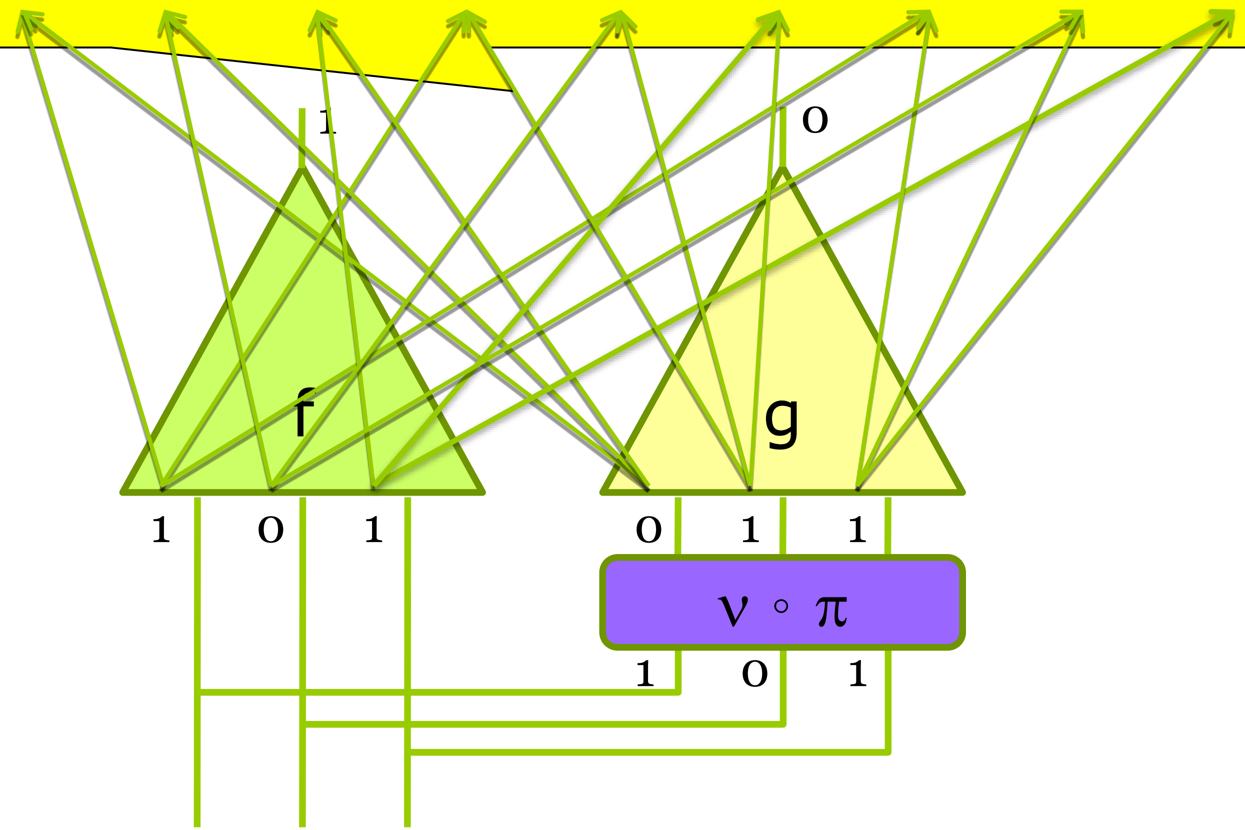
- Observation



NP-Equivalence
Conflict-based Learning

- Learnt clause generation

$(a_{11} \lor b_{12} \lor a_{13} \lor b_{21} \lor a_{22} \lor b_{23} \lor b_{31} \lor a_{32} \lor b_{33})$



NP-Equivalence
Conflict-based Learning

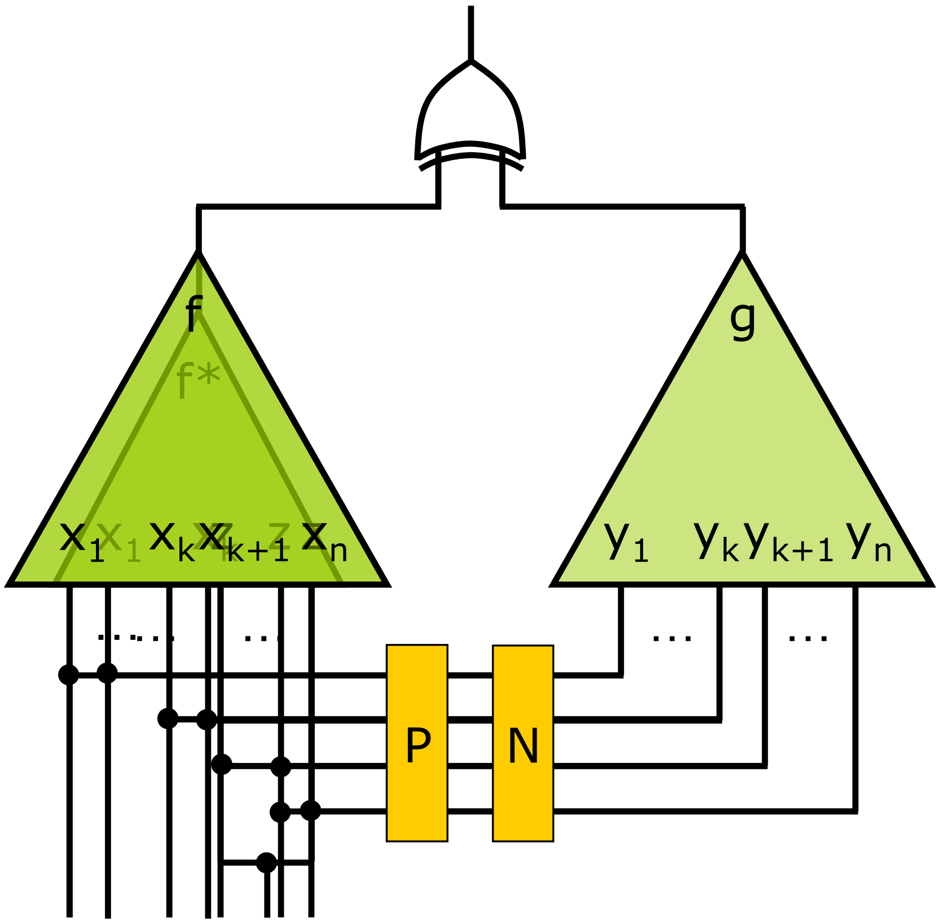
- Proposition:
  If $f(u) \neq g(v)$ with $v = \nu \circ \pi(u)$ for some $\nu \circ \pi$ satisfying $\Phi^{\langle i \rangle}$, then the learned clause $\bigvee_{ij} l_{ij}$ for literals
  $l_{ij} = (v_i \neq u_j)\ ?\ a_{ij} : b_{ij}$
  excludes from $\Phi^{\langle i \rangle}$ the mappings $\{\nu' \circ \pi' \mid \nu' \circ \pi'(u) = \nu \circ \pi(u)\}$

- Proposition:
  The learned clause prunes n! infeasible mappings

- Proposition:
  The refinement process $\Phi^{\langle 0 \rangle}, \Phi^{\langle 1 \rangle}, \ldots, \Phi^{\langle k \rangle}$ is bounded by 2^{2n} iterations

NP-Equivalence Abstraction

- Abstract Boolean matching
  - Abstract $f(x_1,\ldots,x_k,x_{k+1},\ldots,x_n)$ to $f(x_1,\ldots,x_k,z,\ldots,z) = f^*(x_1,\ldots,x_k,z)$
  - Match $g(y_1,\ldots,y_n)$ against $f^*(x_1,\ldots,x_k,z)$
  - Infeasible matching solutions of f* and g are also infeasible for f and g



NP-Equivalence Abstraction

- Abstract Boolean matching
  - Similar matrix representation of negation/permutation

$$
\begin{array}{c|c}
 & \begin{array}{ccccccc} x_1^* & \neg x_1^* & \cdots & x_k^* & \neg x_k^* & z & \neg z \end{array} \\
\begin{array}{c} y_1 \\ y_2 \\ \vdots \\ y_n \end{array} &
\begin{pmatrix}
a_{11} & b_{11} & \cdots & a_{1k} & b_{1k} & a_{1(k+1)} & b_{1(k+1)} \\
a_{21} & b_{21} & \cdots & a_{2k} & b_{2k} & a_{2(k+1)} & b_{2(k+1)} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
a_{n1} & b_{n1} & \cdots & a_{nk} & b_{nk} & a_{n(k+1)} & b_{n(k+1)}
\end{pmatrix}
\end{array}
\quad \sum = 1
$$

$\sum = 1$

- Similar cardinality constraints, except for allowing multiple y-variables mapped to z

NP-Equivalence Abstraction

- Used for preprocessing
- Information learned for abstract model is valid for concrete model
- Simplified matching in reduced Boolean space

P-Equivalence
Conflict-based Learning

- Proposition:

  If $f(u) \neq g(v)$ with $v = \pi(u)$ for some π satisfying $\Phi^{\langle i \rangle}$, then the learned clause $\bigvee_{ij} l_{ij}$ for literals

  $l_{ij} = (v_i{=}0 \text{ and } u_j{=}1)\ ?\ a_{ij} : \varnothing$

  excludes from $\Phi^{\langle i \rangle}$ the mappings $\{\pi' \mid \pi'(u) = \pi(u)\}$

P-Equivalence Abstraction

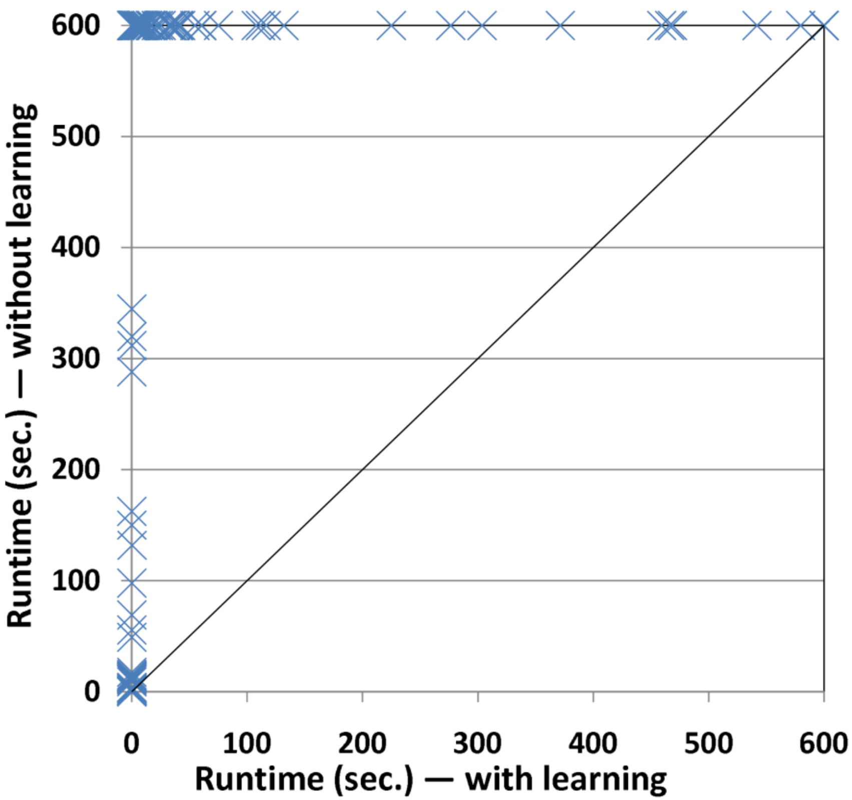
- Abstraction enforces search in biased truth assignments and makes learning strong
  - For f^* having k support variables, a learned clause converted back to the concrete model consists of at most $(k-1)(n-k+1)$ literals

Practical Evaluation

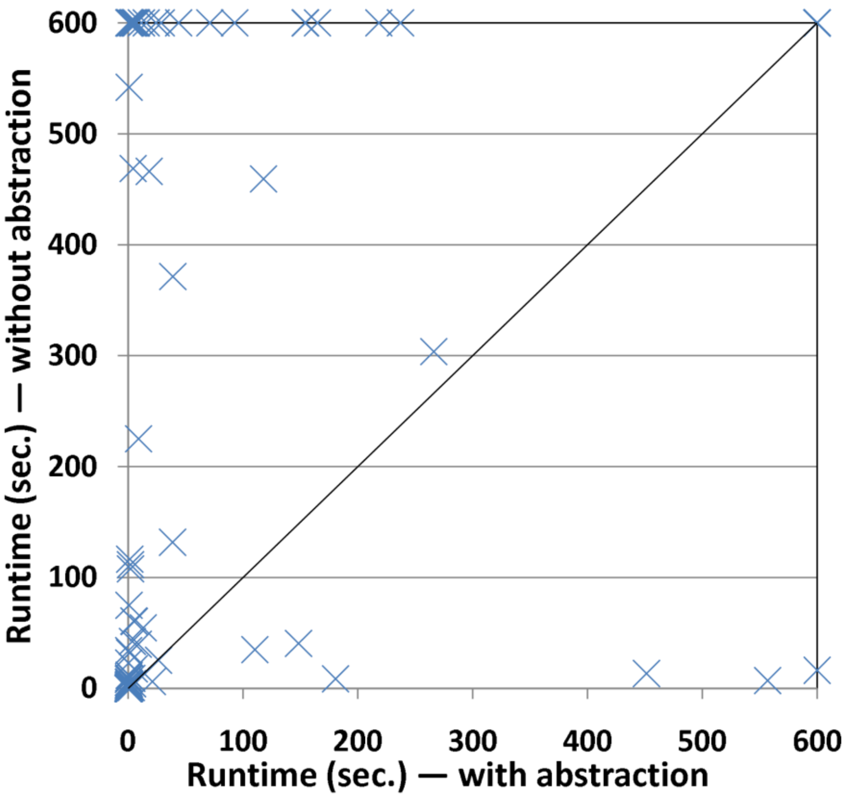
- BooM implemented in ABC using MiniSAT
- A function is matched against its synthesized, and input-permuted/negated version
  - Match individual output functions of MCNC, ISCAS, ITC benchmark circuits
    - 717 functions with 10~39 support variables and 15~2160 AIG nodes
  - Time-limit 600 seconds
  - Baseline preprocessing exploits symmetry, unateness, and simulation for initial matching

Practical Evaluation

Learning



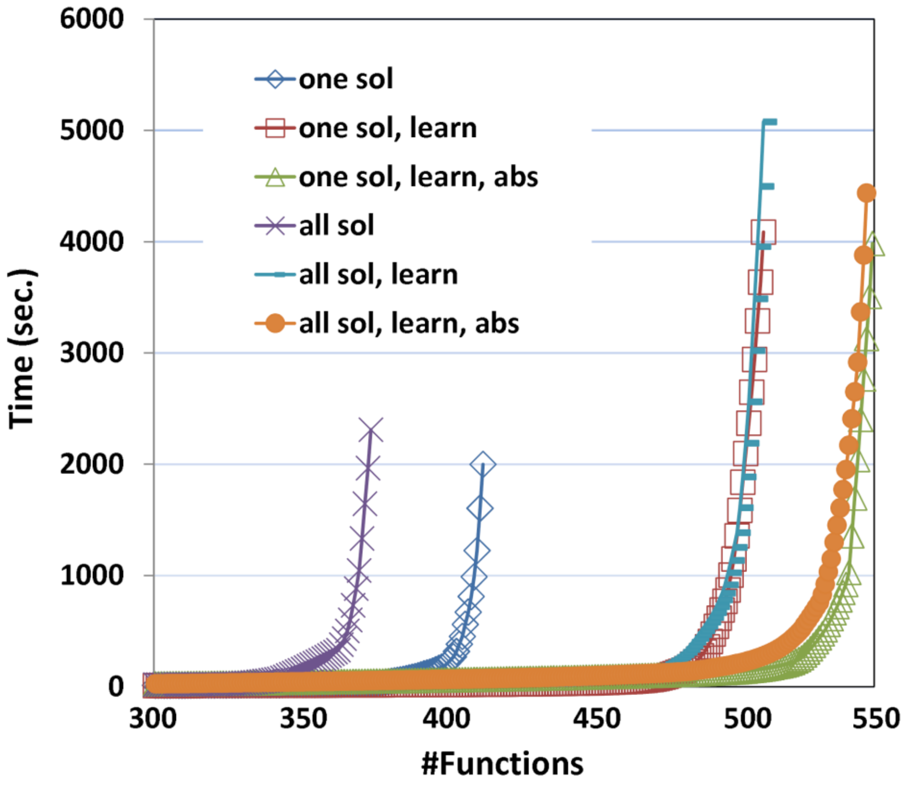
Abstraction



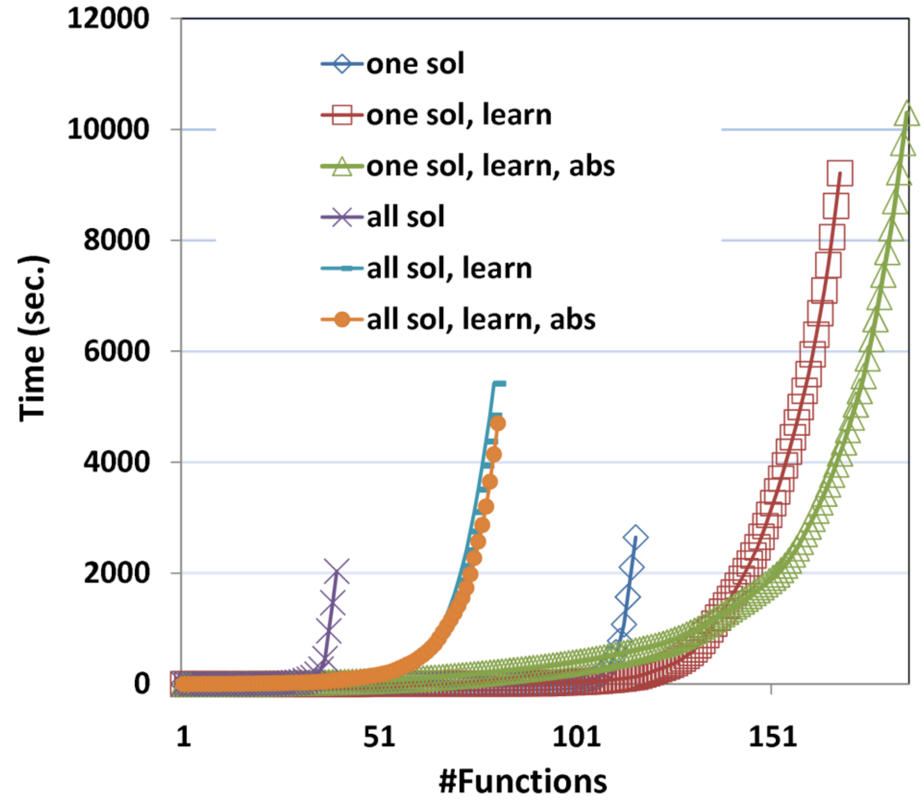
(P-equivalence; find all matches)

Practical Evaluation

P-equivalence

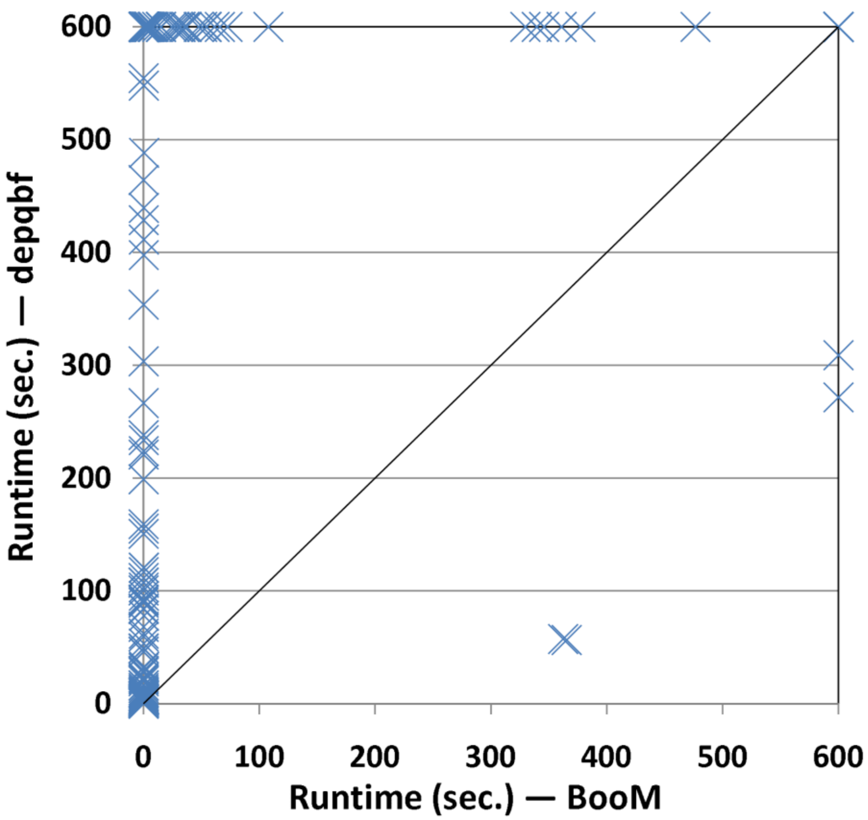


NP-equivalence



Practical Evaluation

BooM vs. DepQBF



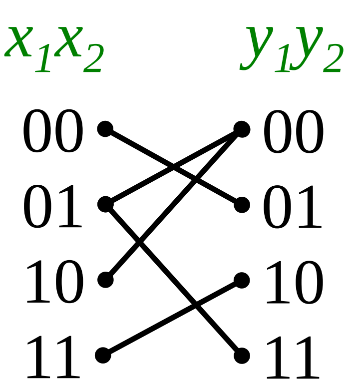
(runtime after same preprocessing; P-equivalence; find one match)

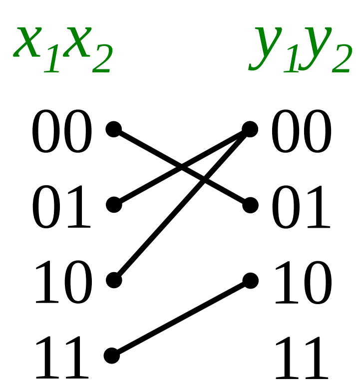
QSAT & Logic Synthesis

Relation Determinization

Relation vs. Function

- 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

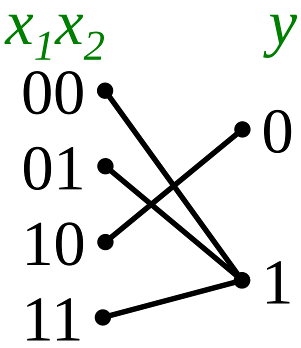




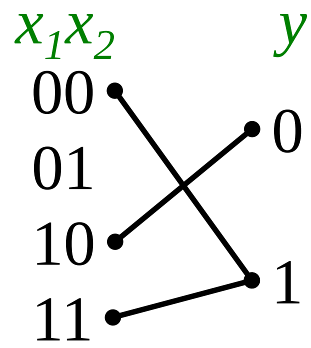
$f_1 = x_1\, x_2$
$f_2 = \neg x_1 \neg x_2$

Relation

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

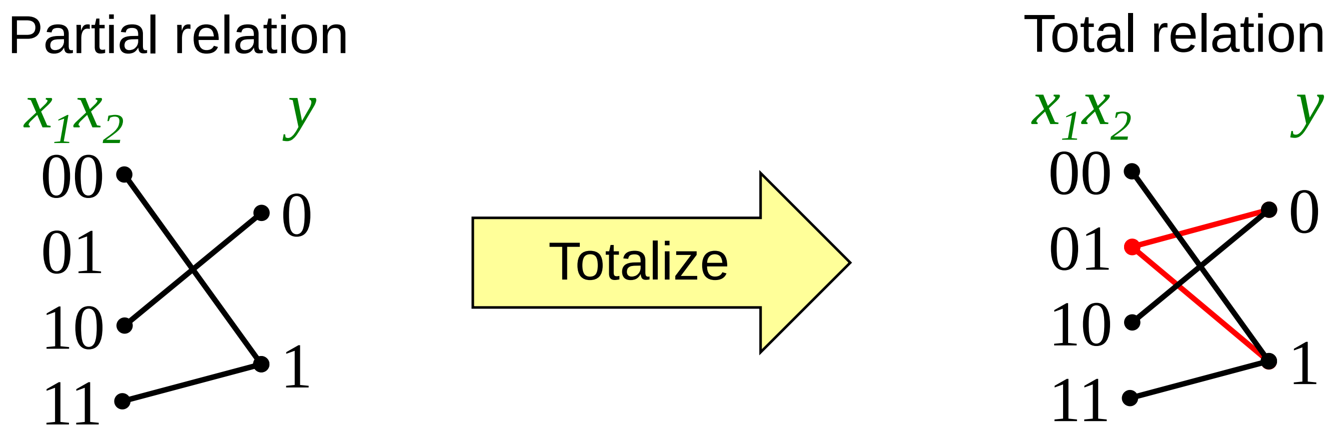


- Partial relation
  - Some input element is not mapped to any output element



Relation

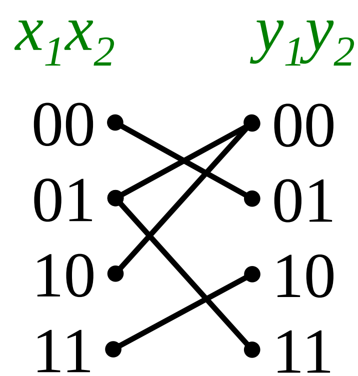
- A partial relation can be **totalized**
  - Assume that the input element not mapped to any output element is a don’t care



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

Motivation

- 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

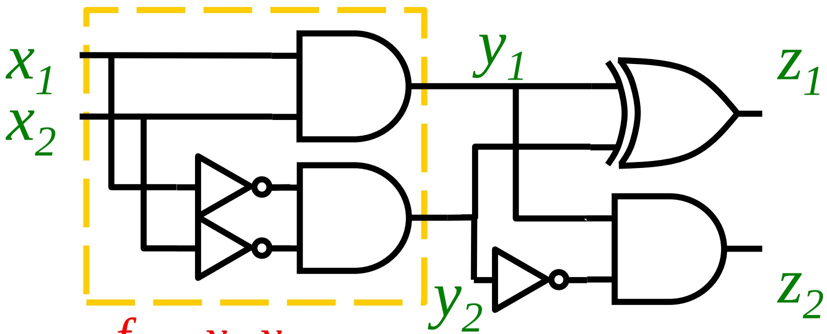


Motivation

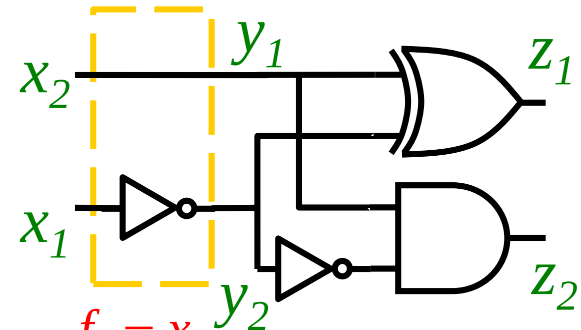
- Relation determinization
  - For hardware implement of a system, we need functions rather than relations
    - Physical realization are deterministic by nature
    - One input stimulus results in one output response
  - To simplify implementation, we can explore the flexibilities described by a relation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

Motivation

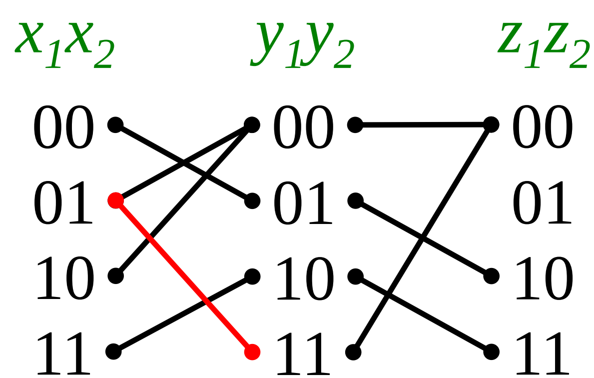
- Example



$f_1 = x_1\, x_2$
$f_2 = \neg\, x_1 \neg\, x_2$



$f_1 = x_2$
$f_2 = \neg\, x_1$



Relation Determinization

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

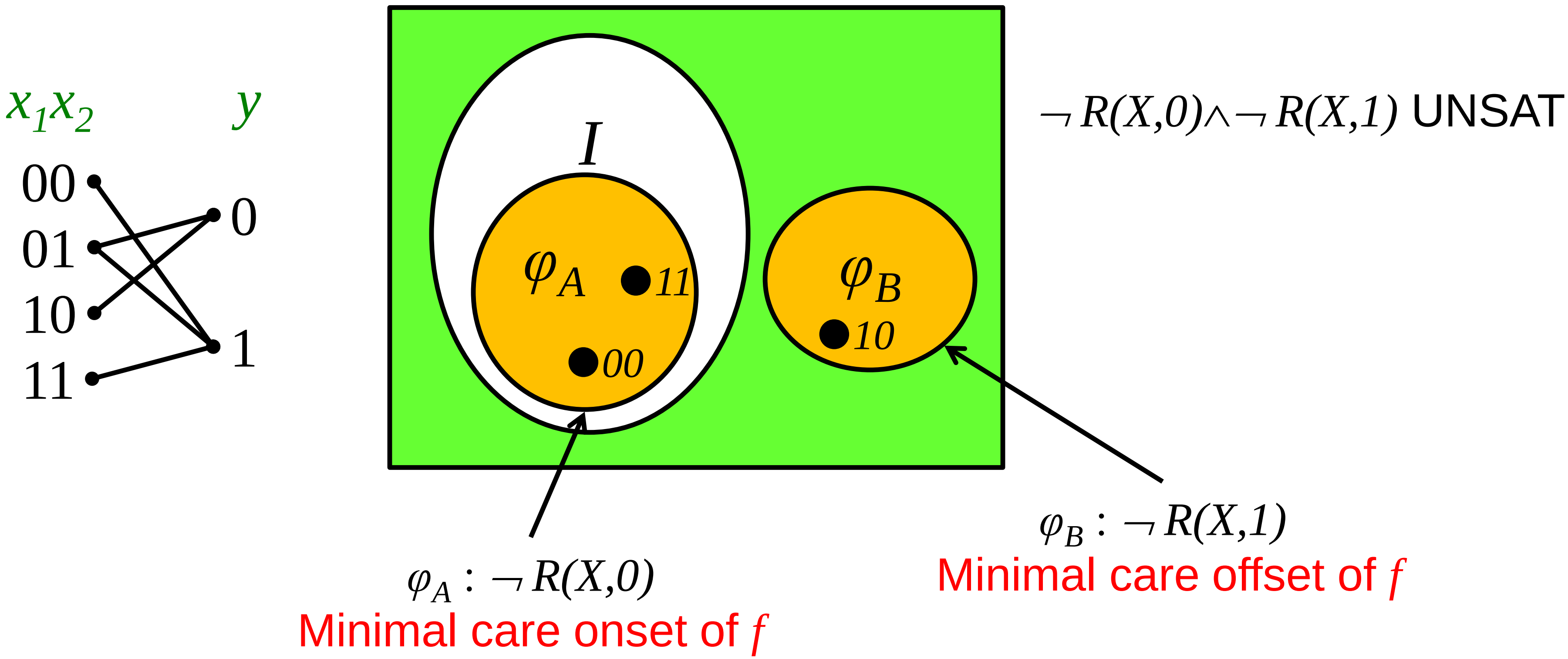
- For a deterministic total relation, we can uniquely extract the corresponding functions

Relation Determinization

- Approaches to relation determinization
  - Iterative method (determinize one output at a time)
    - BDD- or SOP-based representation
      - Not scalable
      - Better optimization
    - AIG representation
      - Focus on scalability with reasonable optimization quality
  - Non-iterative method (determinize all ouputs at once)
    - QBF solving

Iterative Relation Determinization

- Single-output relation
  - For a single-output total relation $R(X, y)$, we derive a function f for variable y using interpolation



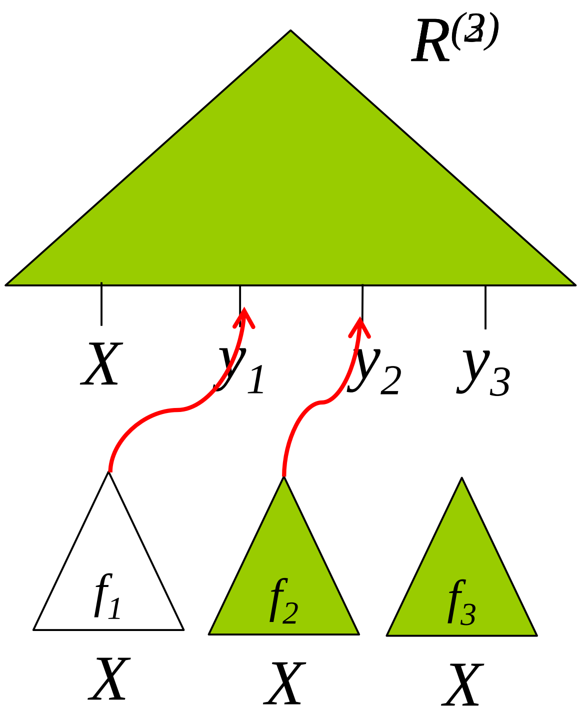
Iterative Relation Determinization

- Multi-output relation
  - Two-phase computation:
    1. Backward reduction
       - Reduce to single-output case

         $R(X, y_1, \ldots, y_n) \rightarrow \exists y_2, \ldots, \exists y_n.\ R(X, y_1, \ldots, y_n)$
    2. Forward substitution
       - Extract functions

Iterative Relation Determinization

- Example



Phase1: (expansion reduction)

$\exists y_3 . R(X, y_1, y_2, y_3) \rightarrow R^{(3)}(X, y_1, y_2)$

$\exists y_2 . R^{(3)}(X, y_1, y_2) \rightarrow R^{(2)}(X, y_1)$

Phase2:

$R^{(2)}(X, y_1) \rightarrow y_1 = f_1(X)$

$R^{(3)}(X, y_1, y_2) \rightarrow R^{(3)}(X, f_1(X), y_2) \rightarrow y_2 = f_2(X)$

$R(X, y_1, y_2, y_3) \rightarrow R(X, f_1(X), f_2(X), y_2) \rightarrow y_3 = f_3(X)$

Non-Iterative Relation Determinization

- Solve QBF

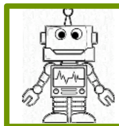
  $\forall x_1, \ldots, \forall x_m, \exists y_1, \ldots, \exists y_n.\ R(x_1, \ldots, x_m, y_1, \ldots, y_n)$

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

Stochastic Boolean Satisfiability

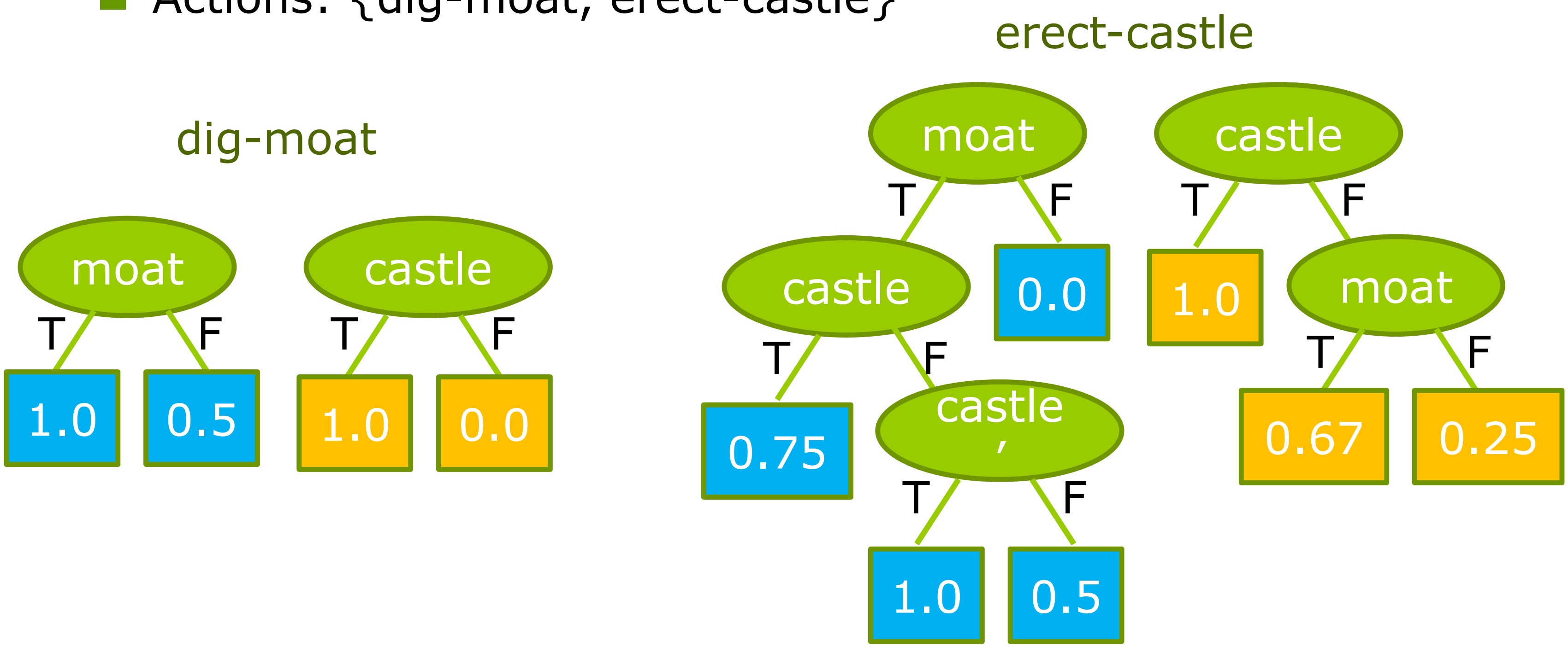
Decision under Uncertainty (Example 1)

- Probabilistic planning: Robot charge [Huang 06]
  - States: $\{S_0, \ldots, S_{15}\}$
    - Initial state: S_0; goal state: S_{15}
  - Actions: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
    - Succeed with prob. 0,8
    - Proceed to its right w.r.t. the intended direction with prob. 0,2

	S_1	S_2	S_3
S_4	S_5	S_6	S_7
S_8	S_9	S_{10}	S_{11}
S_{12}	S_{13}	S_{14}	

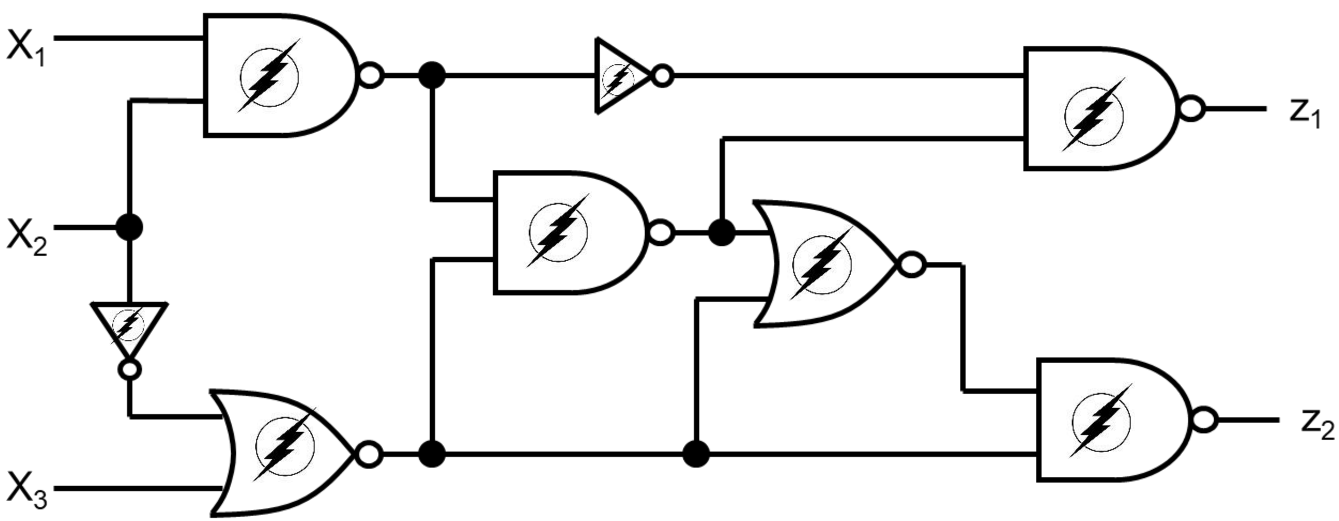
Decision under Uncertainty (Example 2)

- Probabilistic planning: Sand-Castle-67 [Majercik, Littman 98]
  - States: (moat, castle) = {(0,0), (0,1), (1,0), (1,1)}
    - Initial state: (0,0); goal states: (0,1), (1,1)
  - Actions: {dig-moat, erect-castle}



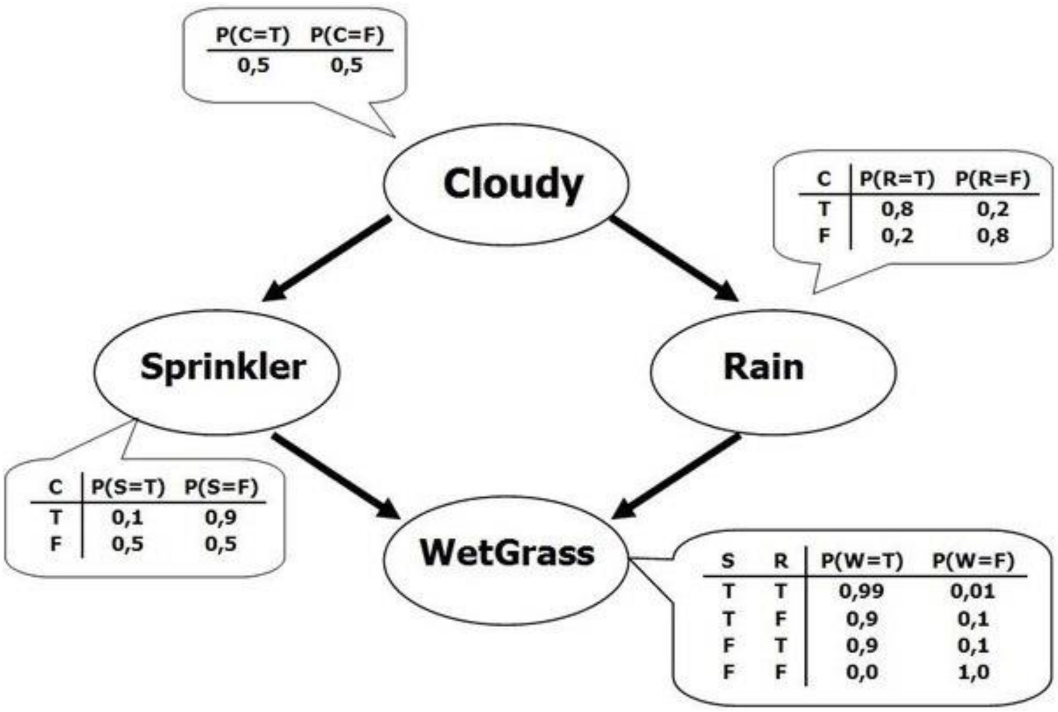
Decision under Uncertainty (Example 3)

- Evaluation of probabilistic circuits [Lee, J 14]
  - Each gate produces correct value under a certain probability
  - Query about the average output error rate, the maximum error rate under some input assignment, etc.



Decision under Uncertainty (Example 4)

- Belief network inference [Dechter 96, Peot 98]
  - BN queries, e.g., belief assessment, most probable explanation, maximum *a posteriori* hypothesis, maximum expected utility

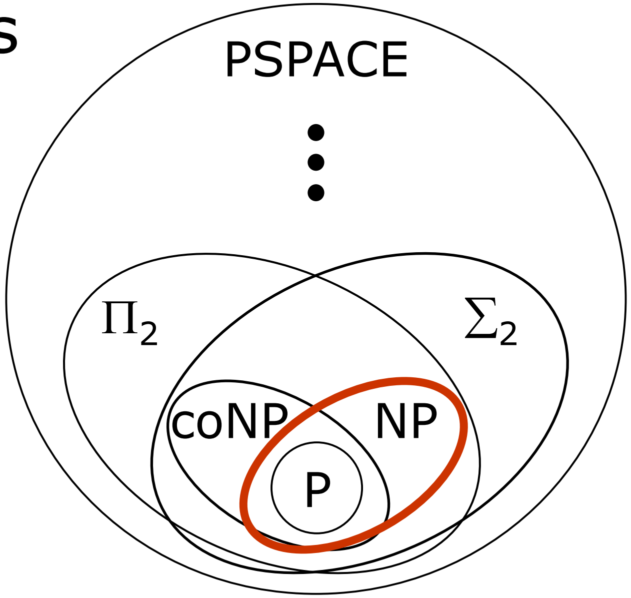


Introduction
The Satisfiability Family

- Boolean satisfiability (SAT)
- Sharp-SAT (#SAT)
- Quantified Boolean satisfiability (QSAT)
- Stochastic Boolean satisfiability (SSAT)

Introduction
The Satisfiability Family – SAT

- The **Boolean satisfiability** (SAT) problem asks whether a given Conjunctive Normal Form (CNF) formula can be satisfied under some assignment to the variables
  - E.g.,
  - $(a+\neg b+c)(a+\neg c)(b+d)(\neg a)$ is satisfiable under $(a,b,c,d)=(0,0,0,1)$
  - $(a+\neg b+c)(a+\neg c)(b)(\neg a)$ is unsatisfiable
- The first known NP-complete problem [Cook 71]

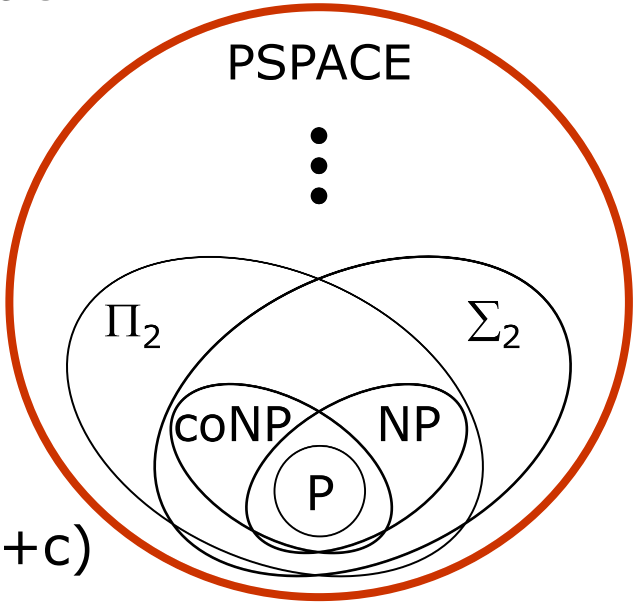


Introduction
The Satisfiability Family – #SAT

- The **#**SAT problem asks the number of satisfying solutions to a given CNF formula
  - E.g., (a+¬b+c)(a+¬c)(b+d)(¬a+b) has five solutions, which are (a,b,c,d) = (0,0,0,1), (1,1,-,-)
  - A #P-complete problem
  - A.k.a. model counting
    - Exact vs. approximate model counting
    - Weighted model counting: variables are weighted under a function w:$var(\phi)$→[0,1]
      - Compute the sum of weights of satisfying assignments of ϕ

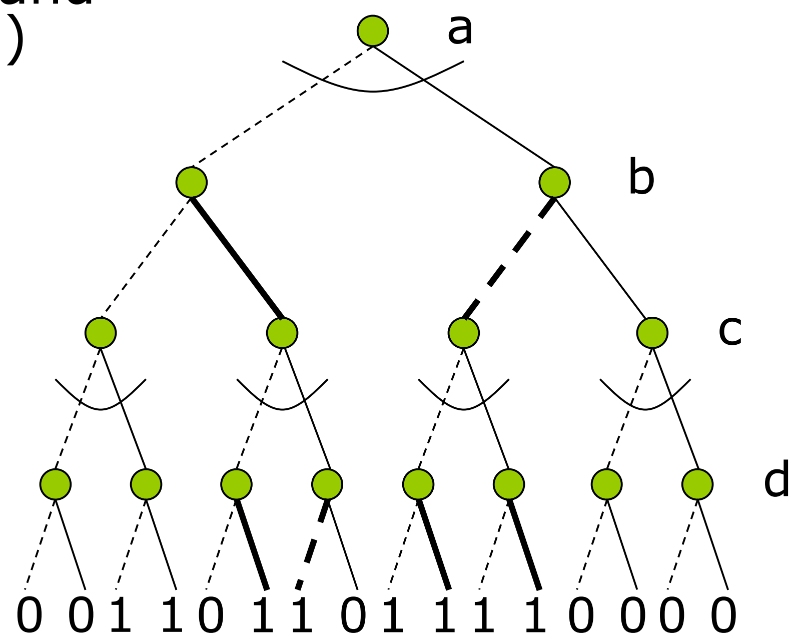
Introduction
The Satisfiability Family – QBF

- A quantified Boolean formula (QBF) is often written in **prenex form** 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 CNF formula
  - E.g., $\forall a\, \exists b\, \forall c\, \exists d.$ $(\neg a+\neg b)(\neg b+\neg c+\neg d)(\neg b+c+d)(a+b+c)$
  - QBF satisfiability is PSPACE-complete



Introduction
The Satisfiability Family – QBF

- A game interpretation of QBF
  - Two-player game played by ∃-player (to satisfy the formula) and ∀-player (to falsity the formula)



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

Skolem functions

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

Introduction
The Satisfiability Family – SSAT

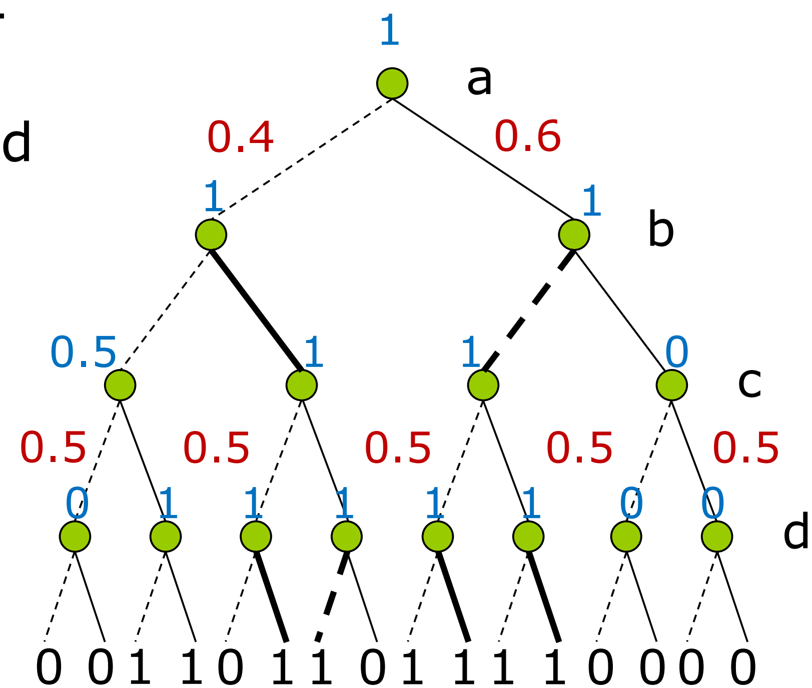
- Syntax of SSAT formula
  $\Phi = Q_1 v_1 \ldots Q_n v_n . \phi(v_1, \ldots, v_n)$
  - Prefix: $Q_1 v_1 \ldots Q_n v_n$ with $Q_i \in \{\exists, \mathcal{R}^{p_i}\}$
    - Randomized quantification $\mathcal{R}^{p_i} v_i$: v_i valuates to `TRUE` with probability p_i
  - Matrix: $\phi(v_1, \ldots, v_n)$ being a quantifier-free propositional formula often in CNF

Introduction
The Satisfiability Family – SSAT

- Semantics of SSAT formula
  $\Phi = Q_1 v_1 \ldots Q_n v_n . \phi(v_1, \ldots, v_n)$
  - Optimization version: Find the maximum SP
  - Decision version: Determine whether SP $\geq \theta$
  - **Satisfying probability (SP):** Expectation of ϕ satisfaction w.r.t. the prefix
    - $\Pr[\top] = 1; \Pr[\bot] = 0$
    - $\Pr[\Phi] = \max\{\Pr[\Phi|_{\neg v}], \Pr[\Phi|_{v}]\}$, for outermost quantification $\exists v$
    - $\Pr[\Phi] = (1-p)\Pr[\Phi|_{\neg v}] + p\Pr[\Phi|_{v}]$, for outermost quantification $\mathcal{R}^p v$

Introduction
Stochastic Boolean Satisfiability

- A game interpretation of SSAT
  - Two-player game played by ∃-player (to maximize the expectation of satisfaction) and $\mathcal{R}$-player (to make random moves)



$\mathcal{R}^{0.6}a\ \exists b\ \mathcal{R}^{0.5}c\ \exists d.$

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

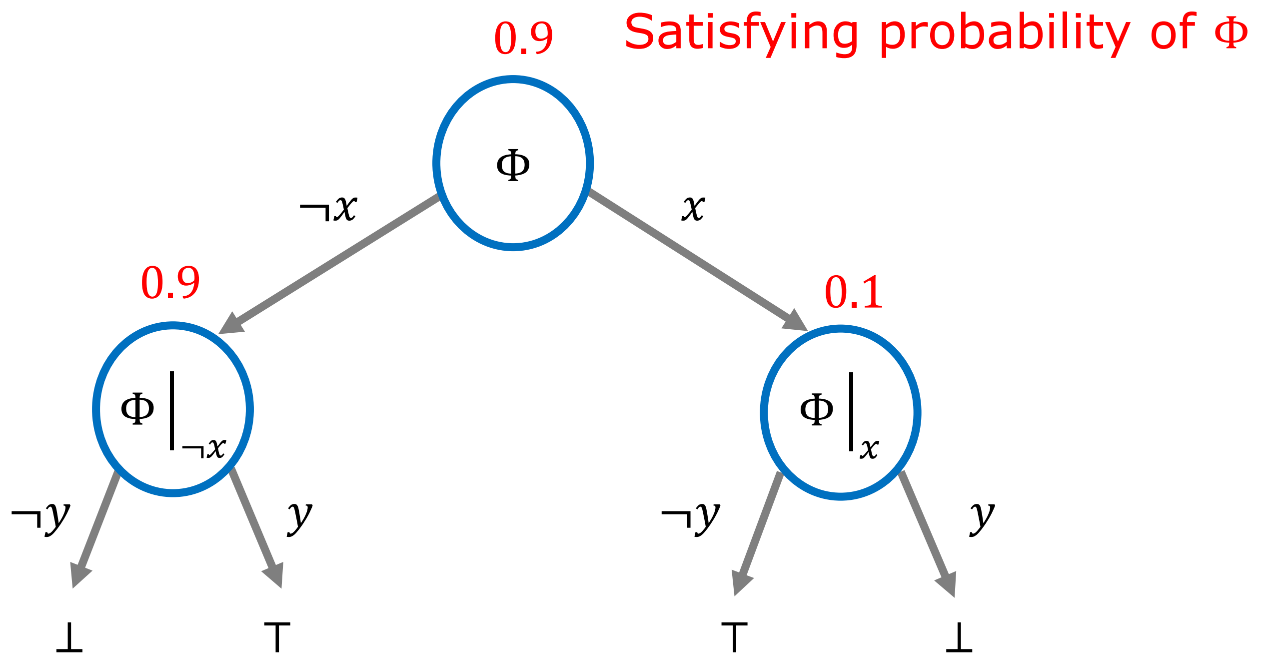
Skolem functions

$\exists F_b(a)\ \exists F_d(a,c)\ \mathcal{R}^{0.6}a\ \mathcal{R}^{0.5}c.$

$(\neg a+\neg F_b)(\neg F_b+\neg c+\neg F_d)(\neg F_b+c+F_d)(a+F_b+c)$

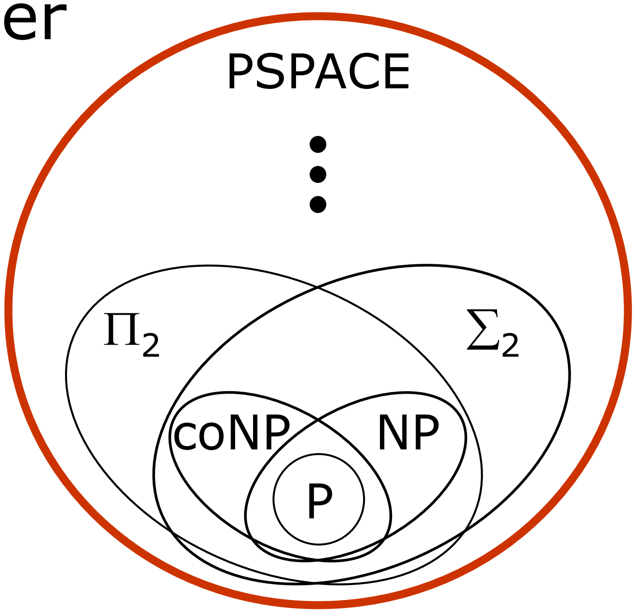
Introduction
The Satisfiability Family – SSAT

- Ex: $\Phi = \exists x \mathcal{R}^{0.9} y. (x \vee y)(\neg x \vee \neg y)$



Introduction
The Satisfiability Family – SSAT

- SSAT is a formalism of games against nature for decision problems under uncertainty [Papadimidriou 85]
- SSAT is PSPACE-complete
- Applications
  - Probabilistic planning
  - Verification of probabilistic circuits
  - Belief network inference
  - Trust management



Introduction
Prior SSAT Methods

- Prior computation methods
  - General SSAT
    - Exact SSAT
      - `DC-SSAT`: divide and conquer, DPLL-style search
      - `ZANDER`: threshold pruning heuristics
    - Approximate SSAT
      - `APPSSAT`: derive upper/lower bounds of satisfying probability
  - E-MAJSAT
    - `MAXPLAN`: pure literal, unit propagation, subproblem memorization
    - `ComPlan`: compilation into d-DNNF
    - `MaxCount`: restricted to $\mathcal{R}^{0.5}$

Introduction
Specialized SSAT of Our Focus

- Random-exist quantified SSAT (RE-SSAT) formula $\Phi = \mathcal{R}X\exists Y.\phi(X,Y)$
  - Counterpart of 2QBF $\Phi = \forall X\exists Y.\phi(X,Y)$

- Exist-random quantified SSAT (ER-SAT, a.k.a. E-MAJSAT) formula $\Phi = \exists X\mathcal{R}Y.\phi(X,Y)$
  - Counterpart of 2QBF $\Phi = \exists X\forall Y.\phi(X,Y)$

Stochastic Boolean Satisfiability

Random-Exist SSAT

RE-SSAT
Main Results

- Exploit weighted model counting to handle randomized quantification
- Use a SAT solver as a plug-in engine for SSAT solving
  - Stand-alone usage of SAT solver and model counter without solver modification
  - Directly benefit from the advancements of SAT solvers and model counters
- Applicable to both exact and approximate RE-SSAT solving

RE-SSAT
Terms and Notations

- Consider $\phi(x_1, x_2, y_1, y_2) = x_1 \wedge (\neg x_2 \vee y_1 \vee y_2)$ with weights $w(x_1) = 0.3$ and $w(x_2) = 0.7$
  - $\tau_1 = x_1 x_2$ is a SAT **minterm**, since $\phi|_{\tau_1}$ can be satisfied by $\mu = y_1 y_2$ → $w(\tau_1) = 0.21$
  - $\tau_1^+ = x_1$ is a SAT **cube** → $w(\tau_1^+) = 0.3$
  - $\tau_2 = \neg x_1 x_2$ is an UNSAT **minterm** since $\phi|_{\tau_2}$ is unsatisfiable → $w(\tau_2) = 0.49$
  - $\tau_2^+ = \neg x_1$ is an UNSAT **cube** → $w(\tau_2^+) = 0.7$
  - The process of expanding τ to τ^+ is called **minterm generalization**

RE-SSAT
Basic Ideas

- Given $\Phi = \mathcal{R}X \exists Y. \phi(X,Y)$, Pr[Φ] equals
  - sum of weights of all SAT minterms, or
  - 1 – sum of weights of all UNSAT minterms
- Collect all SAT and/or UNSAT minterms with minterm generation into cubes
  - SAT: minimal hitting set
  - UNSAT: minimal UNSAT core
- Compute sum of weights of collected cubes
  - Complement the collected cubes into a CNF formula
  - Apply weighted model counting once (needed to cope with the potential non-disjointness between cubes)

RE-SSAT
Procedure for Solving RE-2SSAT

SolveRESSAT

input: $\Phi = \text{Я} X \exists Y.\phi(X, Y)$ and a runtime limit $\texttt{TO}$
output: Upper and lower bounds (P_U, P_L) of satisfying prob.
begin
$\psi(X) := \top$;
$C_\top := \emptyset$;
$C_\bot := \emptyset$;
while $\texttt{SAT}(\psi) = \top \wedge runtime < \texttt{TO}$
  $\tau := \psi$.model;
  if $\texttt{SAT}(\phi|_\tau) = \top$
    $\tau^+ := \mathit{MinimalSatisfying}(\phi, \tau)$;
    $C_\top := C_\top \cup \{\tau^+\}$;
  else $//\texttt{SAT}(\phi|_\tau) = \bot$
    $\tau^+ := \mathit{MinimalConflicting}(\phi, \tau)$;
    $C_\bot := C_\bot \cup \{\tau^+\}$;
  $\psi := \psi \wedge \neg\tau^+$;
return $(1 - \mathit{ComputeWeight}(C_\bot), \mathit{ComputeWeight}(C_\top))$;
end

Matrix solver

Selection solver

If ψ is satisfiable

τ is a SAT minterm

SAT generalization

τ is a UNSAT minterm

UNSAT generalization

Block τ^+ from ψ

Compute weight

RE-SSAT
Example

- $\Phi = \mathcal{R}^{0.5} a, b, c, d \exists x, y, z. \phi$
- $\phi = (a \vee b \vee c \vee x)(a \vee b \vee c \vee \neg x)(\neg a \vee \neg b \vee \neg d \vee y)(\neg a \vee \neg b \vee \neg d \vee \neg y)(\neg a \vee b \vee \neg d \vee z)(\neg a \vee b \vee \neg d \vee \neg z)$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$$\exists x, y, z. \phi(a, b, c, d)$$

	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$\psi(a, b, c, d)$$

SAT cubes:
UNSAT cubes:

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$$\exists x, y, z. \phi(a, b, c, d)$$

	00	01	11	10
00	v	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$\psi(a, b, c, d)$$

SAT cubes:
UNSAT cubes:

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$$\exists x, y, z. \phi(a, b, c, d)$$

	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	1	1	1
10	1	1	1	1

$$\psi(a, b, c, d)$$

SAT cubes:
UNSAT cubes: $\neg a \neg b \neg c$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$\exists x, y, z.\, \phi(a, b, c, d)$

	00	01	11	10
00	0	v	1	1
01	0	1	1	1
11	1	1	1	1
10	1	1	1	1

$\psi(a, b, c, d)$

SAT cubes:
UNSAT cubes: $\neg a \neg b \neg c$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$\exists x, y, z. \phi(a, b, c, d)$

	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

$\psi(a, b, c, d)$

SAT cubes: $\neg ab$
UNSAT cubes: $\neg a \neg b \neg c$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$\exists x, y, z. \phi(a, b, c, d)$

	00	01	11	10
00	0	0	v	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

$\psi(a, b, c, d)$

SAT cubes: $\neg ab$
UNSAT cubes: $\neg a \neg b \neg c$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$$\exists x, y, z.\, \phi(a, b, c, d)$$

	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	0	1	1
10	1	0	0	0

$$\psi(a, b, c, d)$$

SAT cubes: $\neg ab \vee a \neg d$
UNSAT cubes: $\neg a \neg b \neg c$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$\exists x, y, z. \phi(a, b, c, d)$

	00	01	11	10
00	0	0	0	0
01	0	0	v	1
11	1	0	1	1
10	1	0	0	0

$\psi(a, b, c, d)$

SAT cubes: $\neg ab \vee a \neg d$

UNSAT cubes: $\neg a \neg b \neg c$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$$\exists x, y, z.\, \phi(a, b, c, d)$$

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	0	0
10	1	0	0	0

$$\psi(a, b, c, d)$$

SAT cubes: $\neg ab \vee a\neg d$
UNSAT cubes: $\neg a \neg b \neg c \vee ad$

RE-SSAT Example (cont’d)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$\exists x, y, z. \phi(a, b, c, d)$

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	v	0	0	0
10	1	0	0	0

$\psi(a, b, c, d)$

SAT cubes: $\neg ab \vee a\neg d$

UNSAT cubes: $\neg a\neg b\neg c \vee ad$

RE-SSAT
Example (cont'd)

	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	1	1	0	0
10	1	1	1	1

$\exists x, y, z. \phi(a, b, c, d)$

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

$\psi(a, b, c, d)$

SAT cubes: $\neg ab \lor a\neg d \lor \neg ac$
UNSAT cubes: $\neg a\neg b\neg c \lor ad$

RE-SSAT
Example (cont'd)

- Complement the collected SAT cubes $\{\neg ab,\ a\neg d,\ \neg ac\}$ into a CNF formula $\psi = (a \vee \neg b)(\neg a \vee d)(a \vee \neg c)$
- Apply weighted model counting on ψ with weights $w(a) = w(b) = w(c) = w(d) = 0.5$ (recall $\Phi = \mathcal{R}^{0.5} a, b, c, d \exists x, y, z. \phi$)
- Obtain satisfying probability of $\Phi = 0.375$

RE-SSAT
Experimental Settings

- SAT solver `MiniSAT` and weight model counter `Cachet` were used
- Computation platform: Xeon 2.1 GHz CPU and 126 GB RAM
  - Timeout limit: 1000 seconds
- Prior methods under comparison
  - `reSSAT`: the proposed algorithm
  - `reSSAT-b`: the proposed alg. w/o minterm-generalization techniques
  - `DC-SSAT`: state-of-the-art SSAT solver [3]

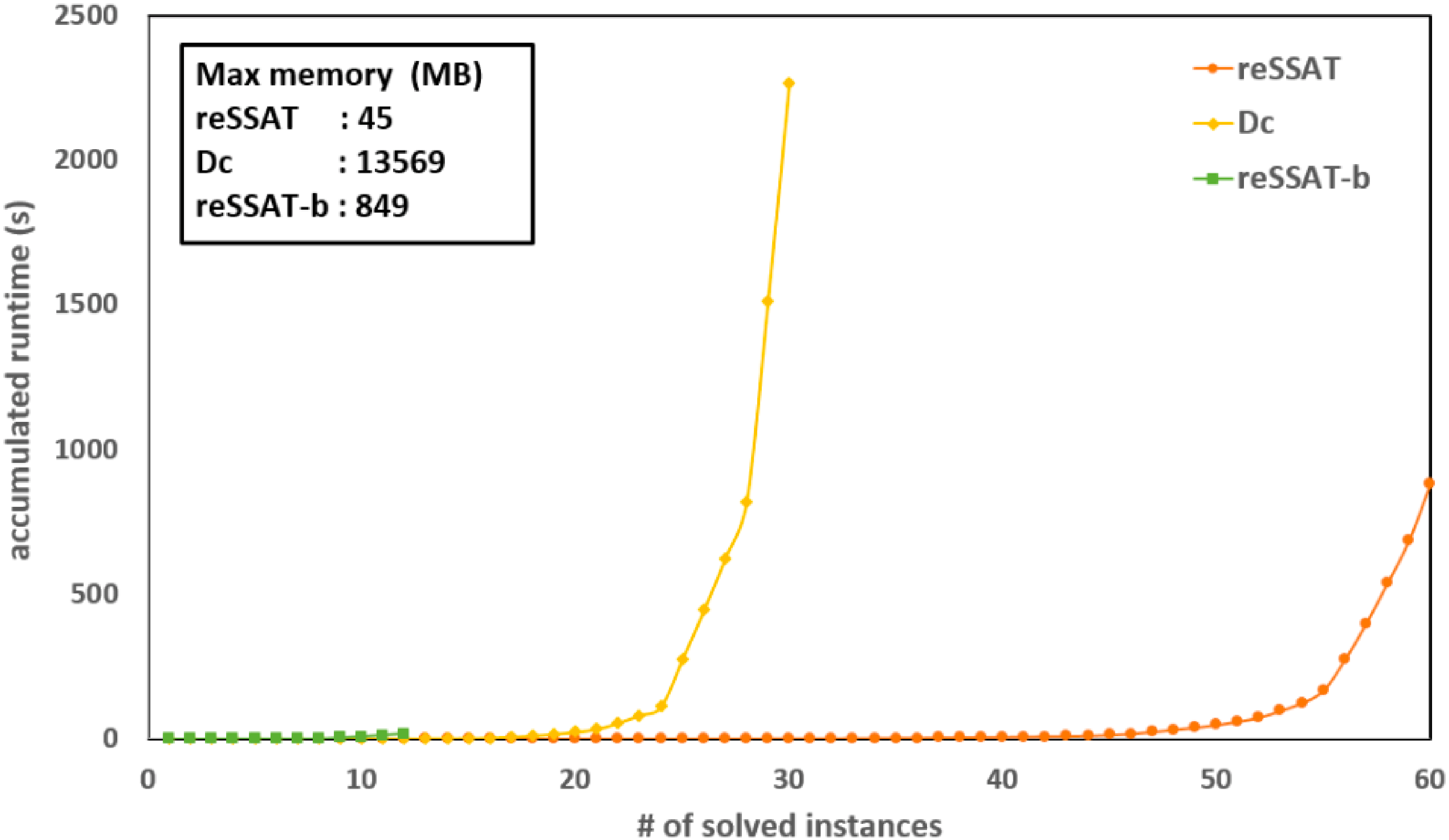
[3] S. Majercik and B. Boots. DCSSAT: A divide-and-conquer approach to solving stochastic satisfiability problems efficiently, 2005

RE-SSAT
Planning Benchmark Experiments

- Converted from 2QBF planning instances of strategic company problem [CEG97]
  - Universal quantifiers in original 2QBFs were changed to randomized ones with probability 0.5
  - The converted RE-2SSAT formulas characterize the winning probabilities of the exist-player of the original QBF games
- 60 formulas from QBFLIB were evaluated
  - `reSSAT-b` solved 12 formulas
  - `DC-SSAT` solved 30 formulas
  - `reSSAT` solve all 60 formulas

[CEG97] M. Cadoli, T. Eiter, and G. Gottlob. Default logic as a query language, 1997.

RE-SSAT
Planning Benchmark Experiments



RE-SSAT
Probabilistic Circuit Experiments

- Obtained in VLSI domain for equivalence checking of probabilistic circuits [LJ14]
  - The formula evaluates the expected difference between a deterministic specification against its probabilistic implementation
  - Encoded as RE-2SSAT formulas

[LJ14] N.-Z. Lee and J.-H. Jiang. Towards formal evaluation and verification of probabilistic design, 2014

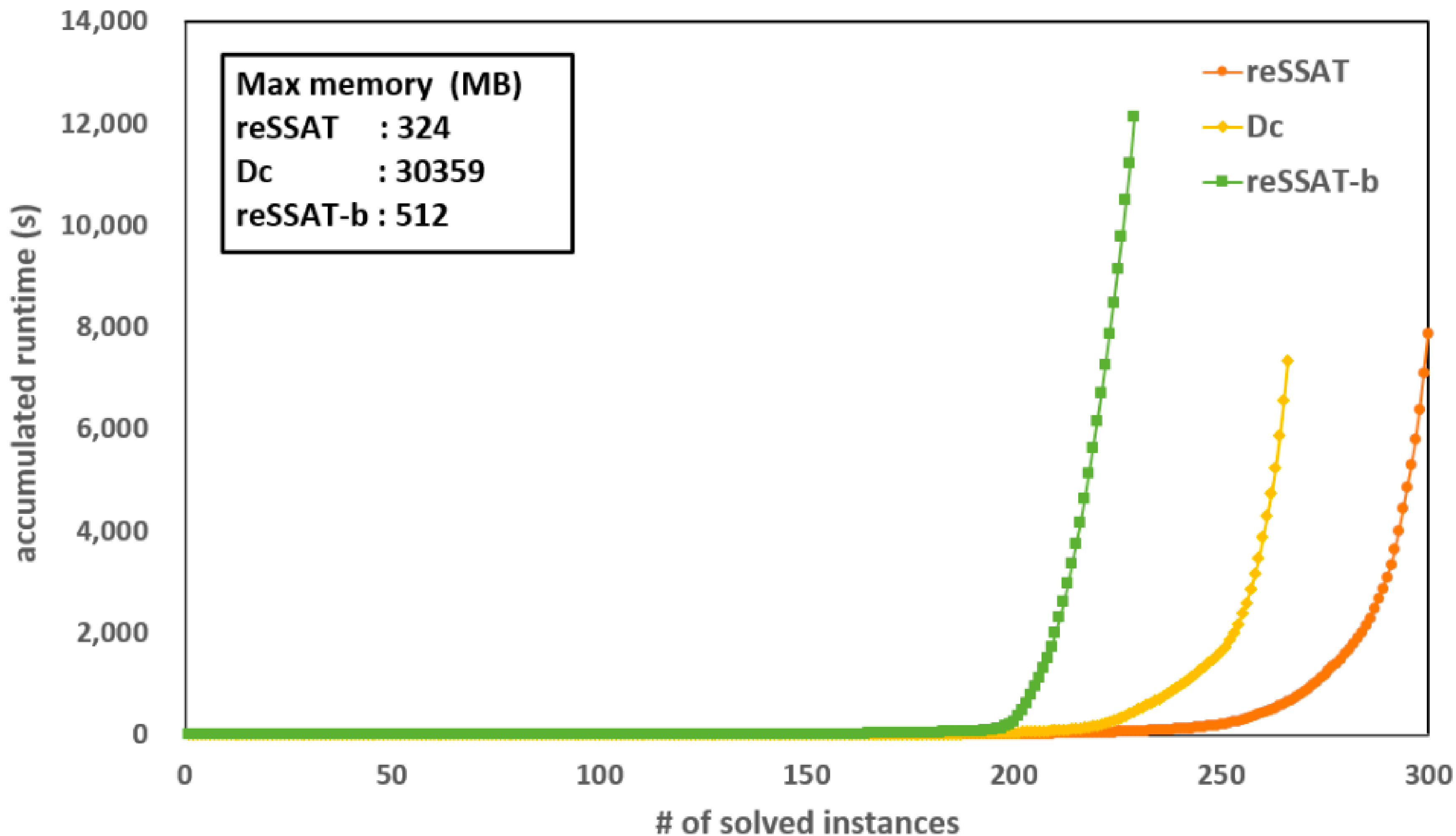
RE-SSAT
Probabilistic Circuit Experiments

		reSSAT (TO=60s)		reSSAT (TO=1000s)		DC-SSAT (TO=1000s)	
circuit	Answer	UB	LB	UB	LB	runtime	Prob.
c432	1.03E-02	1.07E-02	4.30E-05	1.05E-02	8.50E-05	TO	TO
c499	1.56E-13	1.56E-13	1.56E-13	1.56E-13	1.56E-13	0.00	1.56E-13
c880	4.18E-02	9.78E-02	3.00E-06	8.18E-02	3.00E-06	TO	TO
c1355	6.41E-02	3.20E-01	0	3.08E-01	0	TO	TO
c1908	7.38E-04	8.83E-04	4.00E-05	7.38E-04	7.90E-05	210.86	7.38E-04
c3540	1.71E-03	1.17E-02	5.03E-04	1.17E-02	1.61E-03	217.42	1.71E-03
c5315	4.64E-01	6.28E-01	0	6.28E-01	0	TO	TO
c7552	2.34E-01	2.35E-01	7.23E-03	2.35E-01	7.23E-03	TO	TO

RE-SSAT
Random k-CNF Experiments

- Used k-CNF with n variables and m clauses
  - k equals 3, 4, 5, 6, 7, 8, and 9
  - n equals 10, 20, 30, 40, and 50
  - $\frac{m}{n}$ equals $k-1$, k, $k+1$, and $k+2$
- Selected 300 formulas whose satisfying probabilities evenly distributed in [0, 1] for fair evaluation

RE-SSAT
Random k-CNF Experiments



RE-SSAT
Summary

- Proposed a new algorithm to solve random-exist SSAT
  - Plug-in SAT solver and model counter without modification
  - Outperform prior methods in runtime and memory efficiency
- Extended to approximate SSAT with upper/lower bound derivation

Stochastic Boolean Satisfiability

Exist-Random SSAT

ER-SSAT
Main Results

- Adopt QBF clause selection technique to ER-SSAT solving for effective search space pruning
- Propose three enhancement techniques
- Applicable to both exact as well as approximate ER-SSAT

ER-SSAT
Naïve Solution

- Given $\Phi = \exists X \mathcal{R} Y. \phi(X, Y)$
  - Search among assignments τ to X
  - Compute $\mathcal{R} Y. \phi(\tau, Y)$ by weighted model counting
  - Find τ^* maximizing $\mathcal{R} Y. \phi(\tau^*, Y)$

- How to effectively prune search space?

ER-SSAT
Clause Selection for QBF Solving

- $X = \{e_1, e_2, e_3\}, Y = \{a_1, a_2, a_3\}, \phi(X,Y) = \bigwedge_{i=1}^{3} C_i$
  - $C_1 = (e_1 \vee a_1 \vee a_2)$
  - $C_2 = (e_1 \vee e_2 \vee a_1 \vee \neg a_3)$
  - $C_3 = (\neg e_2 \vee \neg e_3 \vee a_2 \vee \neg a_3)$
  - $S = \{s_1, s_2, s_3\}$
    - $\psi(X,S) = (s_1 \equiv \neg e_1) \wedge (s_2 \equiv \neg e_1 \wedge \neg e_2) \wedge (s_3 \equiv e_2 \wedge e_3)$
    - $s_i = \top$ iff C_i is *selected*, i.e., not satisfied by the assignment on X variables [JM15]
    - E.g., $(e_1 = \bot, e_2 = \bot, e_3 = \bot) \rightarrow (s_1 = \top, s_2 = \top)$
- Prune search space by preventing selection of a superset of the current clause set

[JM15] M. Janota and J. Marques-Silva. Solving QBF by clause selection, 2015.

ER-SSAT
Clause Containment Learning (1/2)

- $\Phi = \exists X \mathcal{R} Y.\, \phi(X, Y)$
- $(\phi(\tau_2, Y) \vDash \phi(\tau_1, Y)) \rightarrow (\Pr[\Phi|_{\tau_2}] \leq \Pr[\Phi|_{\tau_1}])$
- Prune assignments that select a superset of selected clauses
- Learning with selection variables
  - $\psi(X, S) \leftarrow \psi(X, S) \wedge C_L$
  - $C_L = \vee \neg s_C$

ER-SSAT
Basic Algorithm

SolveEMAJSAT-basic

input: $\Phi = \exists X \reflectbox{R} Y.\phi(X, Y)$
output: $\Pr[\Phi]$
begin
$\quad \psi(X, S) := (\bigwedge_{C \in \phi}(s_C \equiv \neg C^X)) \wedge (\bigwedge_{\text{pure } l:\mathtt{var}(l) \in X} l)$;
$\quad \mathtt{prob} := 0$;
$\quad$ **while** $\mathtt{SAT}(\psi) = \top$
$\qquad \tau :=$ the found model of ψ for variables in X;
$\qquad$ **if** $\mathtt{SAT}(\phi|_\tau) = \top$
$\qquad\quad \mathtt{prob} := \max\{\mathtt{prob}, \mathtt{WeightModelCount}(\reflectbox{R} Y.\phi|_\tau)\}$;
$\qquad\quad C_L := \bigvee_{C \in \phi|_\tau} \neg s_C$;
$\qquad$ **else** $//\mathtt{SAT}(\phi|_\tau) = \bot$
$\qquad\quad C_L := \mathtt{MinimalConflicting}(\phi, \tau)$;
$\qquad \psi := \psi \wedge C_L$;
$\quad$ **return** $\mathtt{prob}$;
end

ER-SSAT
Example

$\exists a, b, c, d, \mathcal{R}^{0.5}x, \mathcal{R}^{0.7}y, \mathcal{R}^{0.9}z.$

C_1: $\big((a \wedge b \wedge c) \rightarrow (x \vee y \vee z)\big)$

C_2: $\big(\neg c \rightarrow (x \vee \neg y)\big)$

C_3: $\big((\neg b \wedge c) \rightarrow (x \vee z)\big)$

C_4: $\big((\neg a \wedge \neg d) \rightarrow (y \vee z)\big)$

	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Current assignment:
Current max value:
Blocking clause:

$$\psi(a, b, c, d) = \top$$

ER-SSAT
Example (cont'd)

$\exists a, b, c, d, \mathcal{R}^{0.5}x, \mathcal{R}^{0.7}y, \mathcal{R}^{0.9}z.$

C_1: $\big((a \wedge b \wedge c) \rightarrow (x \vee y \vee z)\big)$

C_2: $\big(\neg \boldsymbol{c} \rightarrow (\boldsymbol{x} \vee \neg \boldsymbol{y})\big)$

C_3: $\big((\neg b \wedge c) \rightarrow (x \vee z)\big)$

C_4: $\big((\neg \boldsymbol{a} \wedge \neg \boldsymbol{d}) \rightarrow (\boldsymbol{y} \vee \boldsymbol{z})\big)$

	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Current assignment: $\neg a \neg b \neg c \neg d$
Current max value: 0.62
Blocking clause: $(c \vee a \vee d)$

$$\psi(a, b, c, d) = \top$$

ER-SSAT
Example (cont'd)

$\exists a, b, c, d, \mathcal{R}^{0.5}x, \mathcal{R}^{0.7}y, \mathcal{R}^{0.9}z.$

C_1: $\big((a \wedge b \wedge c) \rightarrow (x \vee y \vee z)\big)$

C_2: $\big(\neg \boldsymbol{c} \rightarrow (\boldsymbol{x} \vee \neg \boldsymbol{y})\big)$

C_3: $\big((\neg b \wedge c) \rightarrow (x \vee z)\big)$

C_4: $\big((\neg a \wedge \neg d) \rightarrow (y \vee z)\big)$

	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Current assignment: $ab\neg c\neg d$
Current max value: 0.65
Blocking clause: (c)

$$\psi = (c \vee a \vee d)$$

ER-SSAT
Example (cont'd)

$\exists a, b, c, d, \mathcal{R}^{0.5}x, \mathcal{R}^{0.7}y, \mathcal{R}^{0.9}z.$

C_1: $\big((a \wedge b \wedge c) \rightarrow (x \vee y \vee z)\big)$

C_2: $\big(\neg c \rightarrow (x \vee \neg y)\big)$

C_3: $\big((\neg \boldsymbol{b} \wedge \boldsymbol{c}) \rightarrow (\boldsymbol{x} \vee \boldsymbol{z})\big)$

C_4: $\big((\neg a \wedge \neg d) \rightarrow (y \vee z)\big)$

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Current assignment: $\neg a \neg b c d$
Current max value: 0.95
Blocking clause: $(b \vee \neg c)$

$$\psi = (c \vee a \vee d)(c)$$

ER-SSAT
Example (cont'd)

$\exists a, b, c, d, \mathcal{R}^{0.5}x, \mathcal{R}^{0.7}y, \mathcal{R}^{0.9}z.$

C_1: $\big((a \wedge b \wedge c) \rightarrow (x \vee y \vee z)\big)$

C_2: $\big(\neg c \rightarrow (x \vee \neg y)\big)$

C_3: $\big((\neg b \wedge c) \rightarrow (x \vee z)\big)$

C_4: $\big((\neg a \wedge \neg d) \rightarrow (y \vee z)\big)$

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	1	1	0

Current assignment: $\neg abcd$
Current max value: 1
Blocking clause: ()

$$\psi = (c \vee a \vee d)(c)$$
$$(b \vee \neg c)$$

ER-SSAT
Example (cont'd)

$\exists a, b, c, d, \mathcal{R}^{0.5} x, \mathcal{R}^{0.7} y, \mathcal{R}^{0.9} z.$

C_1: $\big((a \wedge b \wedge c) \rightarrow (x \vee y \vee z)\big)$

C_2: $\big(\neg c \rightarrow (x \vee \neg y)\big)$

C_3: $\big((\neg b \wedge c) \rightarrow (x \vee z)\big)$

C_4: $\big((\neg a \wedge \neg d) \rightarrow (y \vee z)\big)$

	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Current assignment:
Current max value: 1
Blocking clause: ()

$$\psi = (c \vee a \vee d)(c)$$
$$(b \vee \neg c)()$$

ER-SSAT
Enhancement Techniques

- Minimal clause selection
  - Select a minimal set of clauses by iterative SAT refinement
- Clause subsumption
  - Precompute subsumption relation and remove selected clauses that are subsumed by other selected clauses
- Partial assignment pruning
  - Discard literals from a learnt clause to obtain an upper bound of satisfying probability

ER-SSAT
Refined Algorithm

SolveEMAJSAT

input: $\Phi = \exists X \reflectbox{A} Y.\phi(X, Y)$
output: $\Pr[\Phi]$
begin
  $\psi(X, S) := (\bigwedge_{C \in \phi}(s_C \equiv \neg C^X)) \wedge (\bigwedge_{\text{pure } l : \mathtt{var}(l) \in X} l)$;
  $\mathtt{prob} := 0$;
  $\mathtt{s\text{-}table} := \mathtt{BuildSubsumeTable}(\phi)$;
  while $\mathtt{SAT}(\psi) = \top$
    $\tau :=$ the found model of ψ for variables in X;
    if $\mathtt{SAT}(\phi|_\tau) = \top$
      $\tau' := \mathtt{SelectMinimalClauses}(\phi, \psi)$;
      $\varphi := \mathtt{RemoveSubsumedClauses}(\phi|_{\tau'}, \mathtt{s\text{-}table})$;
      $\mathtt{prob} := \max\{\mathtt{prob}, \mathtt{WeightModelCount}(\reflectbox{A} Y.\varphi)\}$;
      $C_S := \bigvee_{C \in \varphi} \neg s_C$;
      $C_L := \mathtt{DiscardLiterals}(\phi, C_S, \mathtt{prob})$;
    else $//\mathtt{SAT}(\phi|_\tau) = \bot$
      $C_L := \mathtt{MinimalConflicting}(\phi, \tau)$;
    $\psi := \psi \wedge C_L$;
  return $\mathtt{prob}$;
end

ER-SSAT
Approximate ER-SSAT

- Can terminate at any time and return the current best solution
  - A lower bound of the satisfying probability
- Keep deriving tighter lower bounds and converge to the exact solution

ER-SSAT
Experimental Setup

- SAT solver `MiniSAT`
- Weight model counter
  - `Cachet`
  - `CUDD`
- Xeon 2.1 GHz CPU and 126 GB RAM
- Competing solvers
  - `erSSAT`: the proposed algorithm
  - `DC-SSAT`: state-of-the-art SSAT solver
  - `ComPlan`: E-MAJSAT solver (based on `c2d`)
  - `MAXCOUNT`: maximum model counter

ER-SSAT
Application Formulas

- QBF-converted formulas
- Conformant probabilistic planning
  - Sand-castle [ML98]
- MaxSat [FRS17]
- Quantitative information flow [FRS17]
- Program synthesis [FRS17]
- Maximum probabilistic eq. checking [LJ14]

S. Majercik and M. Littman. MAXPLAN: A new approach to probabilistic planning, 1998.

D. Fremont, M. Rabe, and S. Seshia. Maximum model counting, 2017.

N.-Z. Lee and J.-H. Jiang. Towards formal evaluation and verification of probabilistic design, 2014.

ER-SSAT Experimental Results (1/2)

benchmark statistics							erSSAT			Dc		Max			c2d
family	formula	#V	#C	$\#E_1$	#R	$\#E_2$	LB	T_1	T_2	Pr	T	LB	CL	T	T
Toilet-A	10_01.3	106	10604	33	10	63	**1.95e-3**	0	27	**1.95e-3**	13	**1.95e-3**	1.00	36	3
	10_01.5	170	10902	55	10	105	**3.91e-3**	19	577	**3.91e-3**	208	**3.91e-3**	1.00	67	5
	10_01.7	234	11200	77	10	147	**7.81e-3**	179	-	-	-	**7.81e-3**	1.00	294	19
	10_05.2	170	11315	110	10	50	**3.13e-2**	565	-	-	-	-	-	-	
	10_05.3	250	12000	165	10	75	**1.56e-2**	0	-	-	-	-	-	-	244
	10_05.4	330	12685	220	10	100	**1.56e-2**	888	-	-	-	-	-	-	-
	10_10.2	290	12840	220	10	60	**1.00**	3	3	-	-	-	-	-	181
Conformant	blocks_enc_2_b4	3043	57130	1248	7	1788	**4.38e-1**	341	-	-	-	-	-	-	-
	cube_c7_ser—23	1479	15164	138	9	1332	**3.38e-1**	620	-	-	-	-	-	-	-
	cube_c7_ser–opt-24	1542	15510	144	9	1389	**3.44e-1**	679	-	-	-	-	-	-	-
	cube_c9_par—10	847	24106	60	10	777	2.90e-1	185	-	-	-	**2.92e-1**	1.00	802	-
	cube_c9_par–opt-11	928	24548	66	10	852	**2.89e-1**	192	-	-	-	-	-	-	-
	emptyroom_e3_ser–20	982	6286	80	6	896	**1.88e-1**	869	-	-	-	-	-	-	-
	ring_r4_ser–opt-11	373	5333	44	9	320	**4.96e-1**	506	-	-	-	4.53e-1	1.00	102	29
Sand-Castle	SC-11	101	201	22	55	24	**9.77e-1**	32	50	**9.77e-1**	0	-	-	-	0
	SC-12	110	219	24	60	26	**9.84e-1**	133	187	**9.84e-1**	0	-	-	-	0
	SC-13	119	237	26	65	28	**9.89e-1**	441	619	**9.89e-1**	0	-	-	-	0
	SC-14	128	255	28	70	30	**9.92e-1**	632	-	**9.92e-1**	1	-	-	-	0
	SC-15	137	273	30	75	32	9.93e-1	979	-	**9.94e-1**	1	-	-	-	1
	SC-16	146	291	32	80	34	9.94e-1	785	-	**9.96e-1**	3	-	-	-	0
	SC-17	155	309	34	85	36	9.94e-1	654	-	**9.97e-1**	6	-	-	-	1
MaxSat	keller4.clq	120	1212	43	15	62	**9.76e-1**	0	0	-	-	9.13e-1	0.82	5	1
QIF	backdoor-2x16-8	200	272	32	32	136	**5.96e-8**	0	-	-	-	**5.96e-8**	1.00	9	1
	backdoor-32-24	147	76	32	32	83	**1.00**	0	0	-	-	1.95e-3	0.82	601	0
	bin-search-16	1448	5825	16	16	1416	1.95e-3	106	-	-	-	**9.85e-1**	0.91	230	-
	CVe-2007-2875	784	1740	32	32	720	**1.00**	2	2	-	-	9.85e-1	0.82	13	342
	pwd-backdoor	400	609	64	64	272	0.00	-	-	-	-	**9.85e-1**	0.99	93	1
	reverse2	333	293	32	32	269	**2.98e-7**	271	-	-	-	-	-	-	2
	reverse	229	293	32	32	165	**5.96e-7**	839	-	-	-	-	-	-	2
PS	ConcreteActService	4836	17866	71	37	4728	0.00	-	-	-	-	**9.60e-1**	0.82	52	-
	IssueServiceImpl	3625	13028	77	29	3519	0.00	-	-	-	-	**9.06e-1**	0.82	34	-
	IterationService	4167	15264	70	34	4063	0.00	-	-	-	-	**9.70e-1**	0.82	47	-
	LoginService	5229	21566	92	27	5110	0.00	-	-	-	-	**9.45e-1**	0.82	56	-
	PhaseService	4167	15264	70	34	4063	0.00	-	-	-	-	**9.70e-1**	0.82	47	-
	ProcessBean	9880	41451	166	39	9675	0.00	-	-	-	-	**9.27e-1**	0.82	126	-
	UserServiceImpl	4019	14657	87	31	3901	0.00	-	-	-	-	**9.22e-1**	0.82	43	-
MPEC	c499(2.34e-1)	217	522	41	2	174	**2.34e-1**	0	0	**2.34e-1**	0	**2.34e-1**	1.00	0	2
	c880(2.34e-1)	451	1167	60	2	389	**1.25e-1**	0	-	-	-	**1.25e-1**	1.00	14	72
	c1355(3.30e-1)	771	2181	41	3	727	**3.30e-1**	0	-	-	-	**3.30e-1**	1.00	41	10
	c1908(2.34e-1)	270	705	33	2	235	**2.34e-1**	23	-	**2.34e-1**	91	1.25e-1	1.00	1	3
	c3540(1.25e-1)	321	807	50	2	269	**1.25e-1**	0	-	**1.25e-1**	92	**1.25e-1**	1.00	2	3
	c5315(7.37e-1)	918	2190	178	10	730	4.14e-1	154	-	-	-	**6.27e-1**	0.82	63	217
	c7552(4.87e-1)	648	1308	207	5	436	**2.34e-1**	0	-	-	-	2.18e-1	0.82	66	5
Maximum memory usage (GB)									2.2		38.6			0.2	4.2

ER-SSAT Experimental Results (2/2)

- Compared to `DCSSAT`
  - Exactly solve or derive the tightest lower bounds when `DCSSAT` solves a formula
  - Derive lower bounds when `DCSSAT` fails
- Compared to `MaxCount`
  - Scale better on QBF-converted and planning
  - Derive tighter lower bounds on circuits
  - Perform worse on QIF and PS
- Derive more tightest lower bounds than `DCSSAT` and `MaxCount` for all formulas

ER-SSAT
Summary

- Propose an algorithm to solve ER-SSAT
  - Clause containment learning
  - Approximate ER-SSAT
  - Exactly solve or derive the tightest bounds when state-of-the-art solvers solve a formula
  - Derive lower bounds when other solvers fail

Summary

- We learned
  - Representations of Boolean functions
  - Boolean satisfiability
  - Quantified Boolean satisfiability
  - Stochastic Boolean satisfiability

- To explore logic synthesis and verification, Berkeley ABC tool
  - https://people.eecs.berkeley.edu/~alanmi/abc/