Formal Equivalence Checking - II

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Dept. of Electrical Engineering
Indian Institute of Technology Bombay, Mumbai
viren@ee.iitb.ac.in

EE 709: Testing & Verification of VLSI Circuits

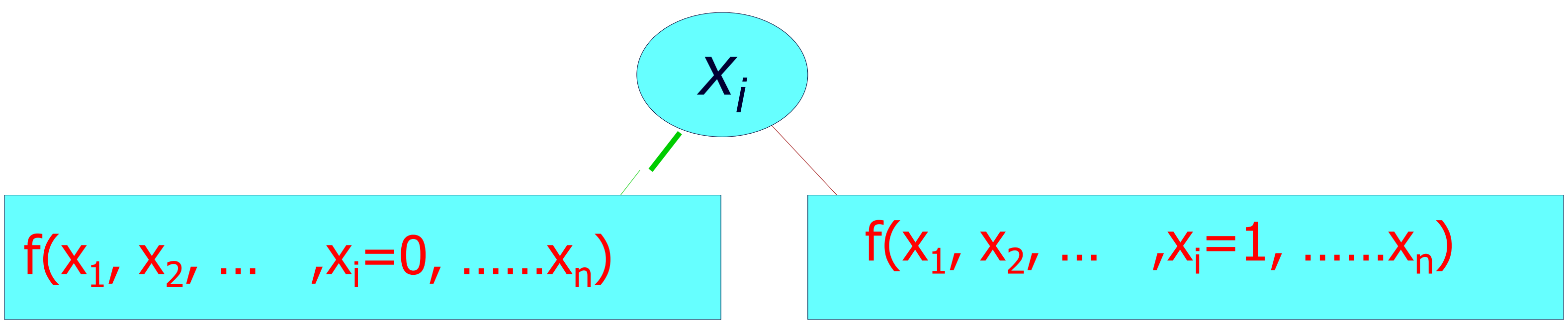
Lecture – 7 (Jan 18, 2012)

Formal Equivalence Checking

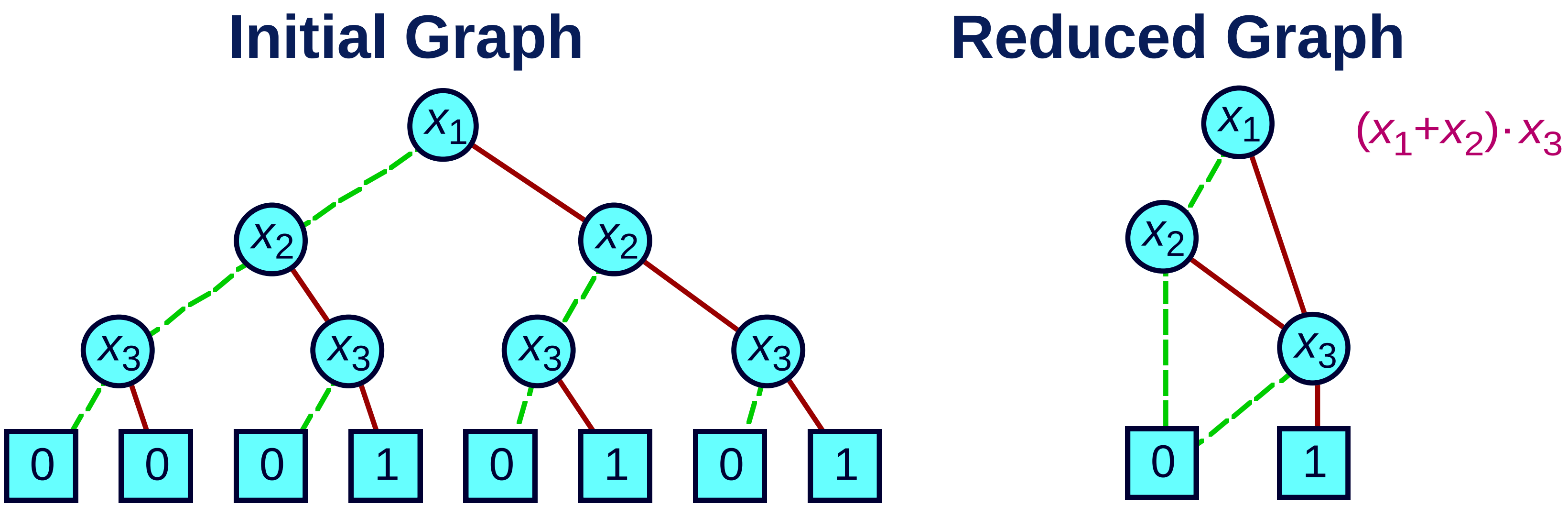
- BDD is canonical form of representation
- Shannon's expansion theorem
- $f(x_1, x_2, ...x_i, ...x_n) =$

  $x_i.f(x_1, x_2, ... ,x_i=1, ...x_n) +$

  $x_i'. f(x_1, x_2, ... ,x_i=0, ...x_n)$



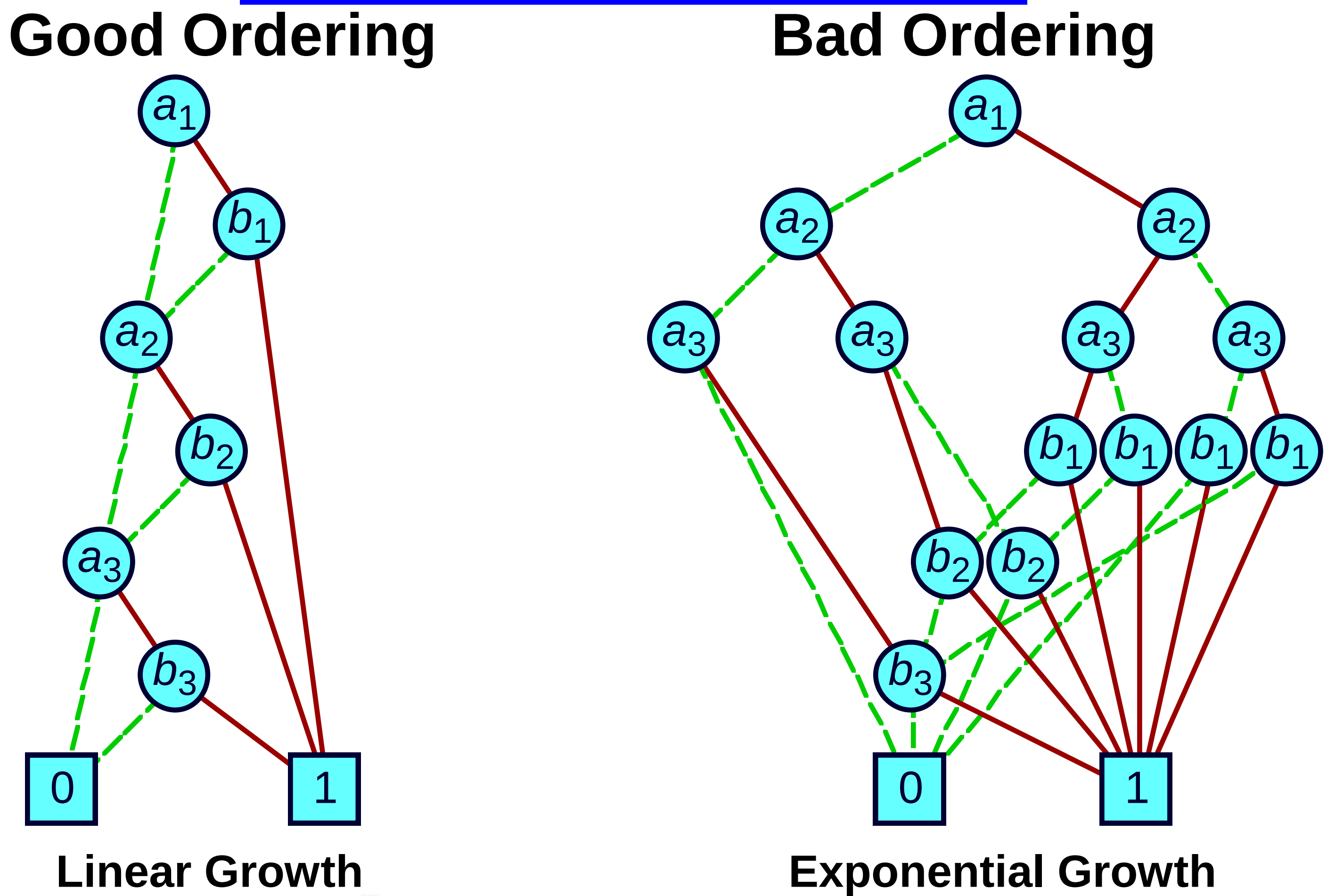
Example OBDD



- Canonical representation of Boolean function
  - ❖ For given variable ordering
  - ➢ Two functions equivalent if and only if graphs isomorphic
    - o Can be tested in linear time
  - ➢ Desirable property: *simplest form is canonical*.

Effect of Variable Ordering

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$$



Sample Function Classes

Function Class	Best	Worst	Ordering Sensitivity
ALU (Add/Sub)	linear	exponential	High
Symmetric	linear	quadratic	None
Multiplication	exponential	exponential	Low

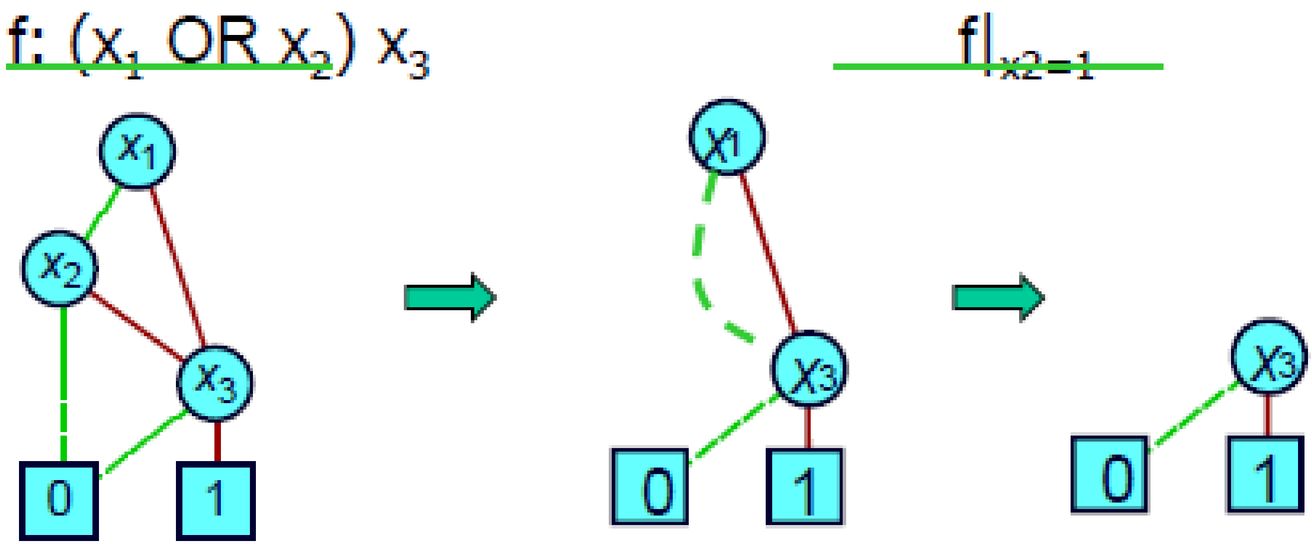
- General Experience
  - Many tasks have reasonable OBDD representations
  - Algorithms remain practical for up to 500,000 node OBDDs
  - Heuristic ordering methods generally satisfactory

ROBDD sizes & variable ordering

- Bad News 💣
  - Finding optimal variable ordering NP-Hard
  - Some functions have exponential BDD size for all orders *e.g.* multiplier
- Good News ☺
  - Many functions/tasks have reasonable size ROBDDs
  - Algorithms remain practical up to 500,000 node OBDDs
  - Heuristic ordering methods generally satisfactory
- What works in Practice ☜
  - Application-specific heuristics e.g. DFS-based ordering for combinational circuits
  - Dynamic ordering based on variable sifting *(R. Rudell)*

Operations with BDD (1/5)

- Restriction: A restriction to a function to x=d, denoted $f|_{x=d}$, where $x \in var(f)$, and $d \in \{0,1\}$, is equal to f after assigning x = d.
- Given BDD of f, deriving BDD of $f|_{x=d}$ is simple



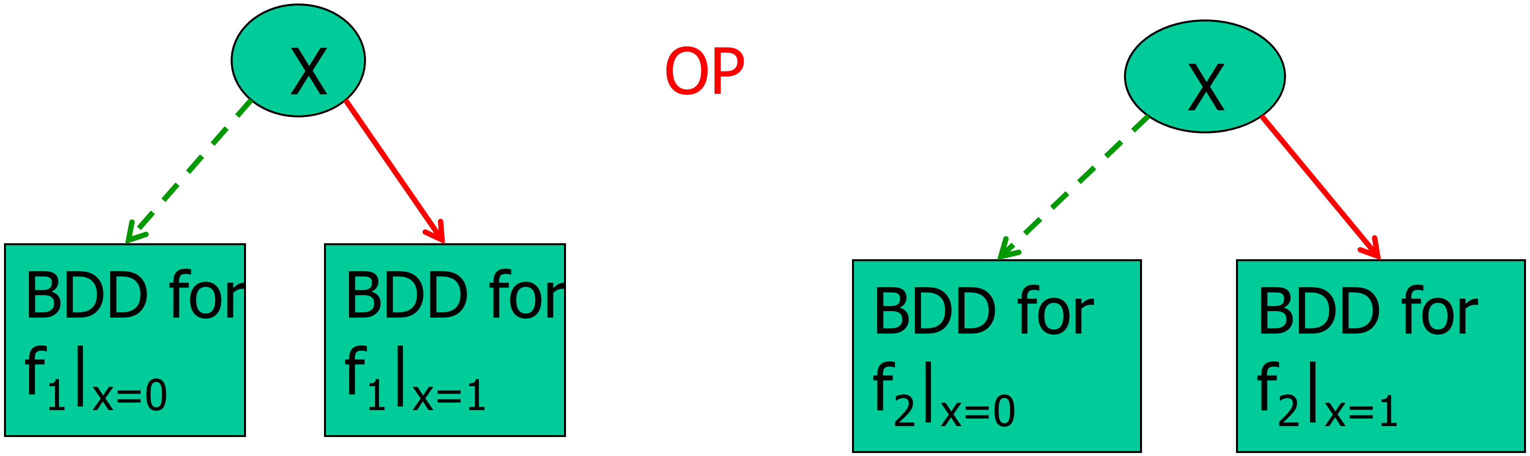
Operations with BDD (2/5)

- Let v_1, v_2 denote root nodes of f_1, f_2 respectively , with $var(v_1) = x_1$ and $var(v_2) = x_2$
- If v_1 and v_2 are leafs, f_1 OP f_2 is a leaf node with value $val(v_1)$ OP $val(v_2)$

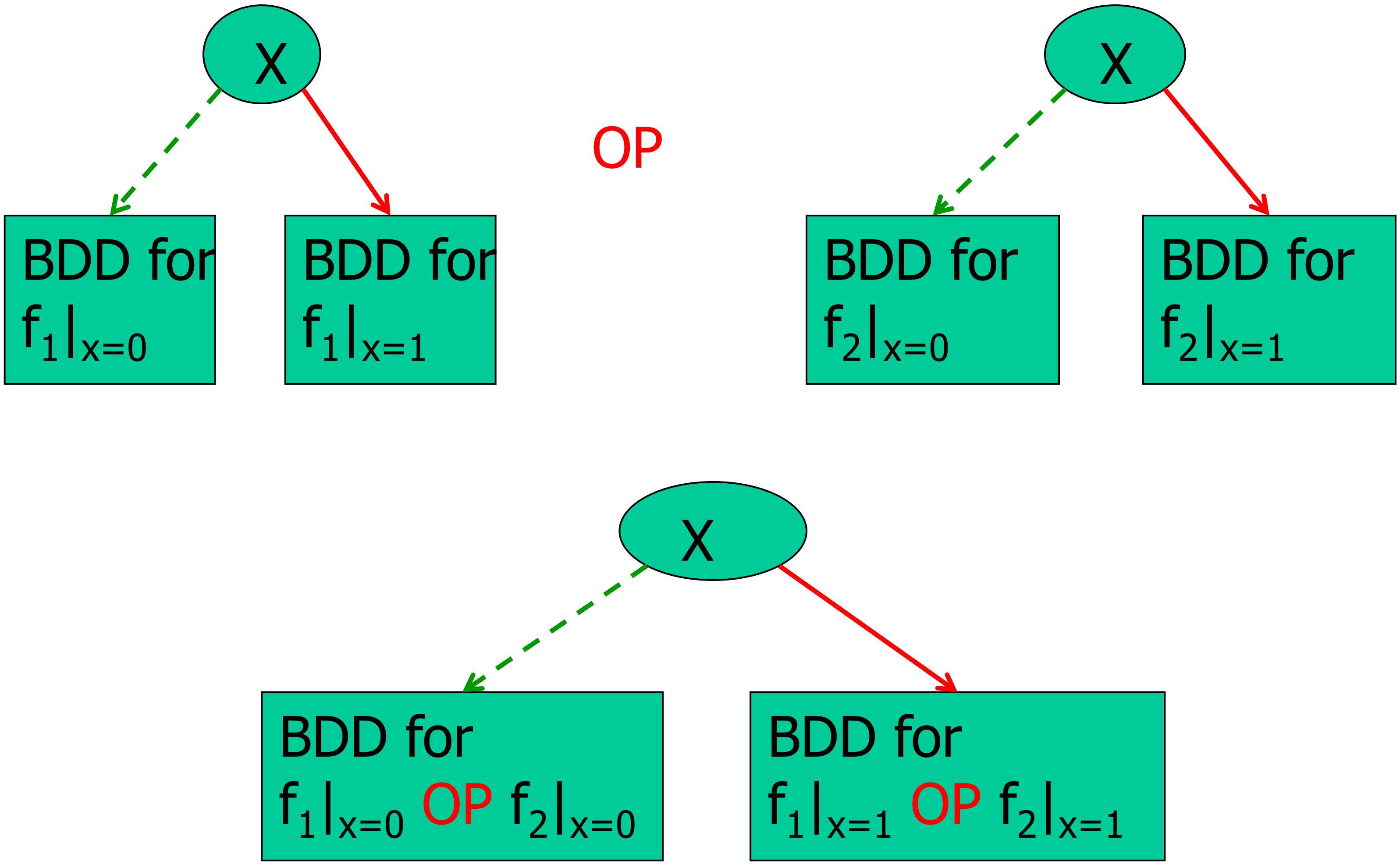
Operations with BDD (3/5)

- If $x_1 = x_2 = x$, apply shanon's expansion

$$f_1 \text{ OP } f_2 = x \,.\, (f_1|_{x=0} \text{ OP } f_2|_{x=0}) + x' \,.\, (f_1|_{x=1} \text{ OP } f_2|_{x=1})$$



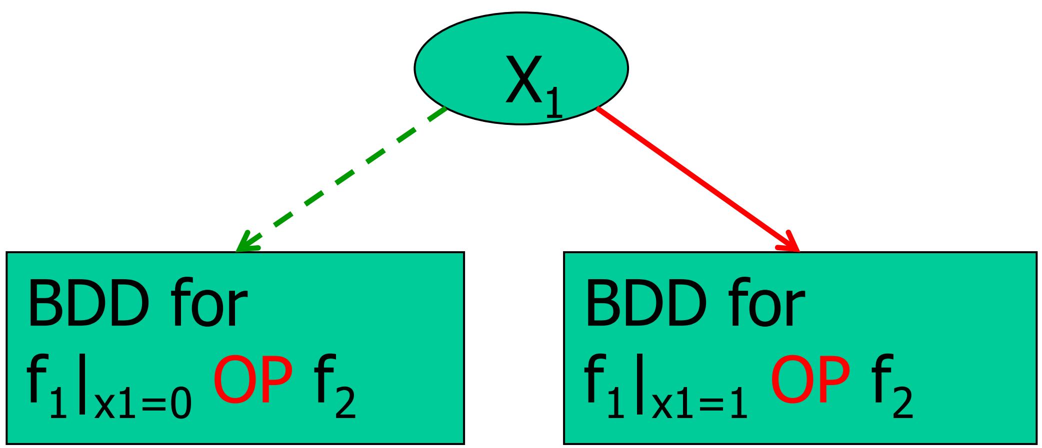
Operations with BDD (4/5)



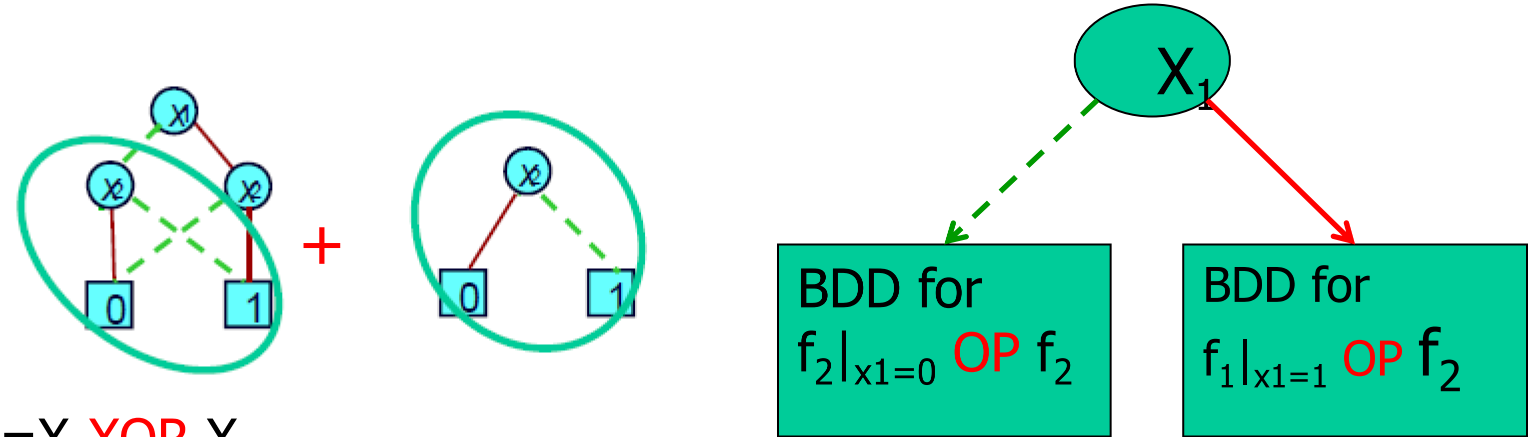
Operations with BDD (5/5)

- Else suppose $x_1 < x_2 = x$, in variable order

$$f_1 \text{ OP } f_2 = x_1 \, (f_1|_{x1=0} \text{ OP } f_2) + x_1' \, (f_1|_{x1=1} \text{ OP } f_2)$$

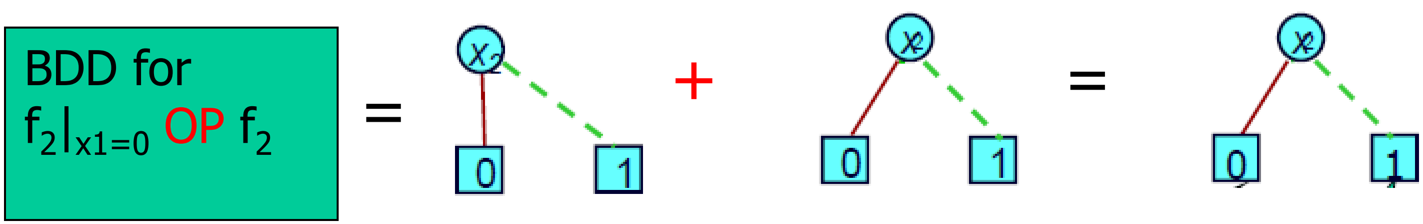


Operations with BDD: Example

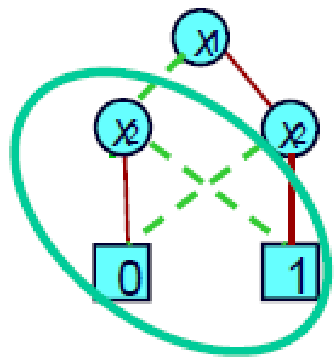


$f_1 = X_1$ XOR X_2

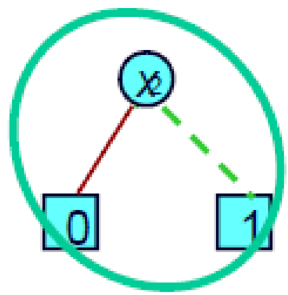
$f_2 = X_2$



Operations with BDD: Example



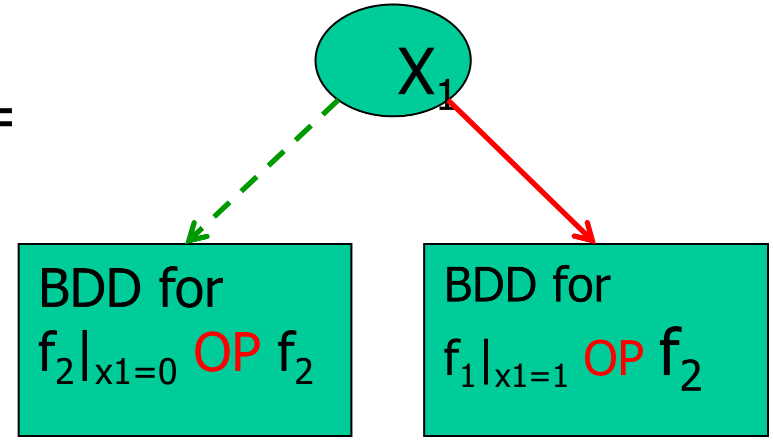
OP



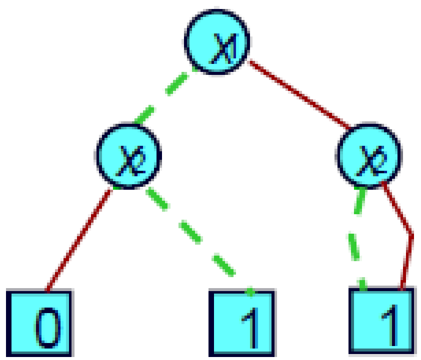
$f_1 = X_1$ XOR X_2

$f_2 = X_2$

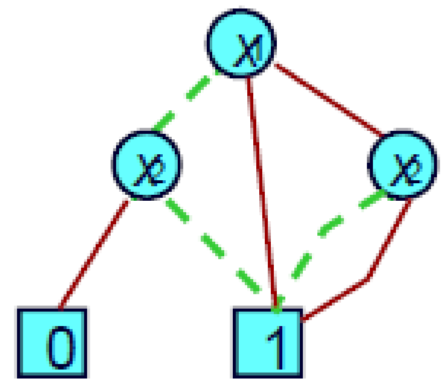
=



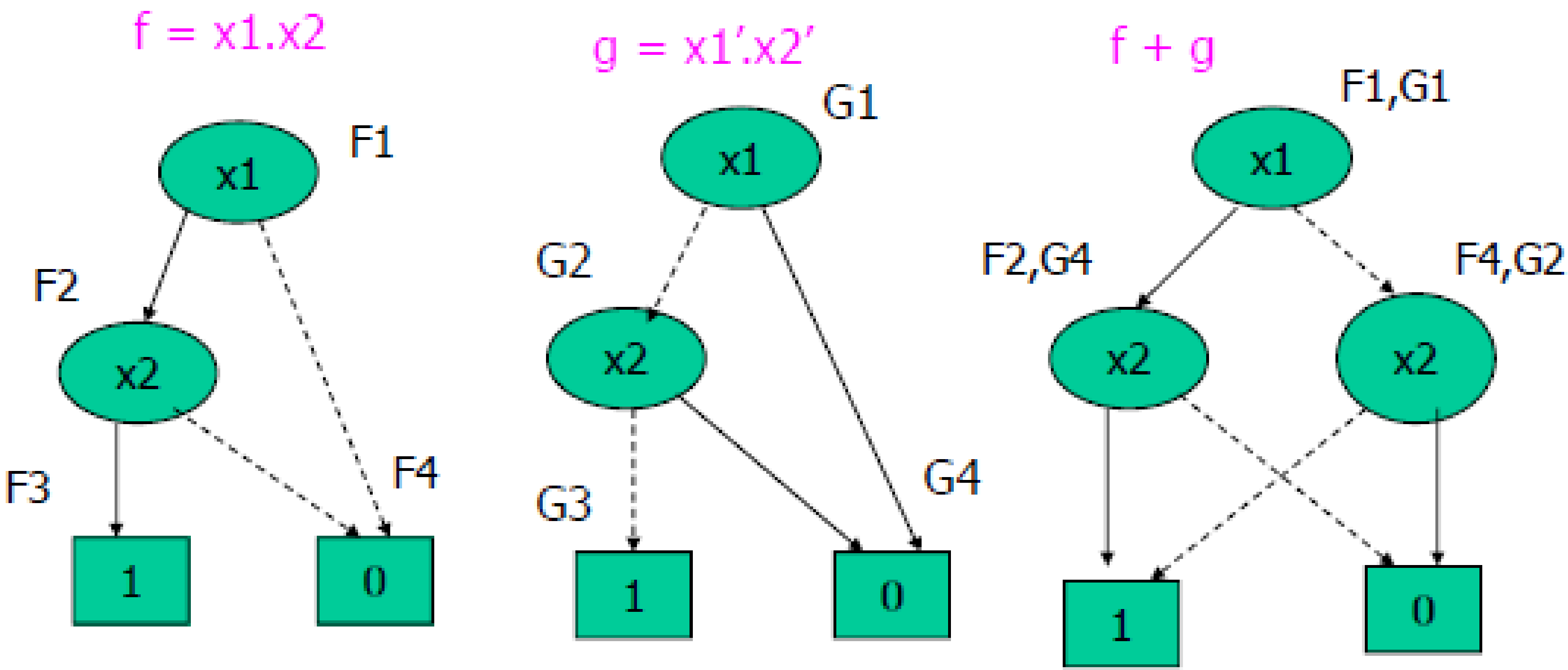
=



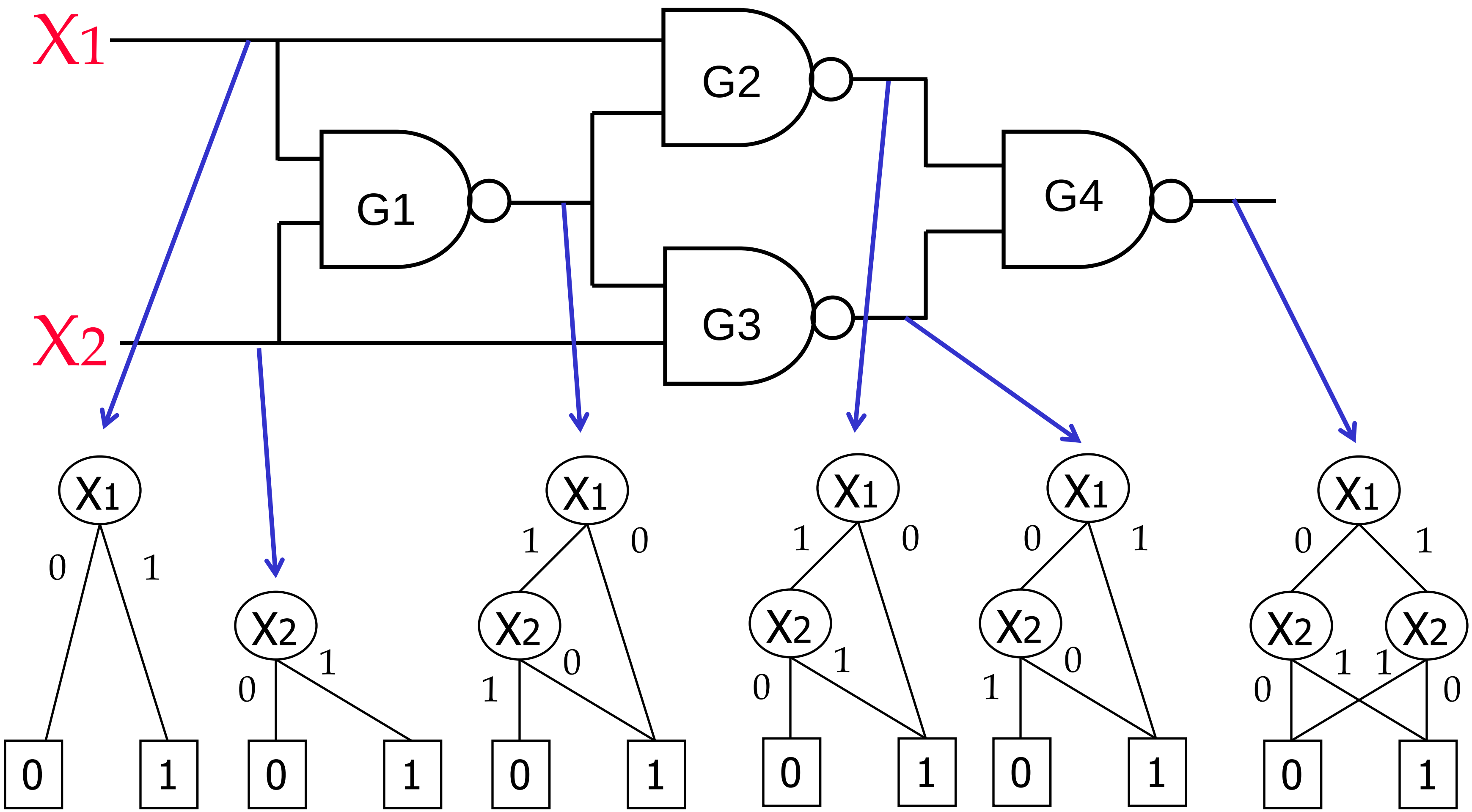
=



Operations with BDD: Example



From circuits to BDD

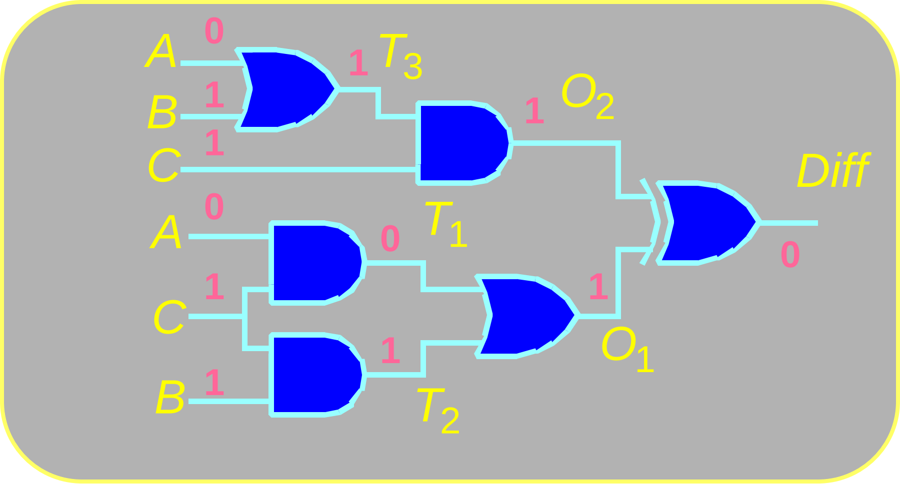


Variants of decision diagrams

- Multiterminal BDDs (MTBDD) – Pseudo Boolean functions $B^n \rightarrow N$, terminal nodes are integers
- Ordered Kronecker FunctionalDecision Diagrams (OKFDD) – uses XOR in OBDDs
- Binary Moment Diagrams (BMD) – good for arithmetic operations and word-level representation
- Zero-suppressed BDD (ZDD) – good for representing sparse sets
- Partitioned OBDDs (POBDD) – highly compact representation which retains most of the features of ROBDDs
- BDD packages –
  - CUDD from Univ. of Colorado, Boulder,
  - CMU BDD package from Carnegie Mellon Univ.
  - In addition, companies like Intel, Fujitsu, Motorola etc. have their own internal BDD packages

Formal Equivalence Checking

- Satisfiability Formulation
  - Search for input assignment giving different outputs
- Branch & Bound
  - Assign input(s)
  - Propagate forced values
  - Backtrack when cannot succeed



- Challenge
  - Must prove all assignments fail
    - Co-NP complete problem
  - Typically explore significant fraction of inputs
  - Exponential time complexity

SAT Problem definition

Given a CNF formula, f :

- A set of variables, *V* *(a,b,c)*
- Conjunction of clauses *(C_1,C_2,C_3)*
- Each clause: disjunction of literals over *V*

Does there exist an assignment of Boolean values to the variables, V which sets at least one literal in each clause to '1' ?

Example : $\underbrace{(a+b+\bar{c})}_{C_1}\underbrace{(\bar{a}+c)}_{C_2}\underbrace{(a+\bar{b}+c)}_{C_3}$ *a = b = c = 1*

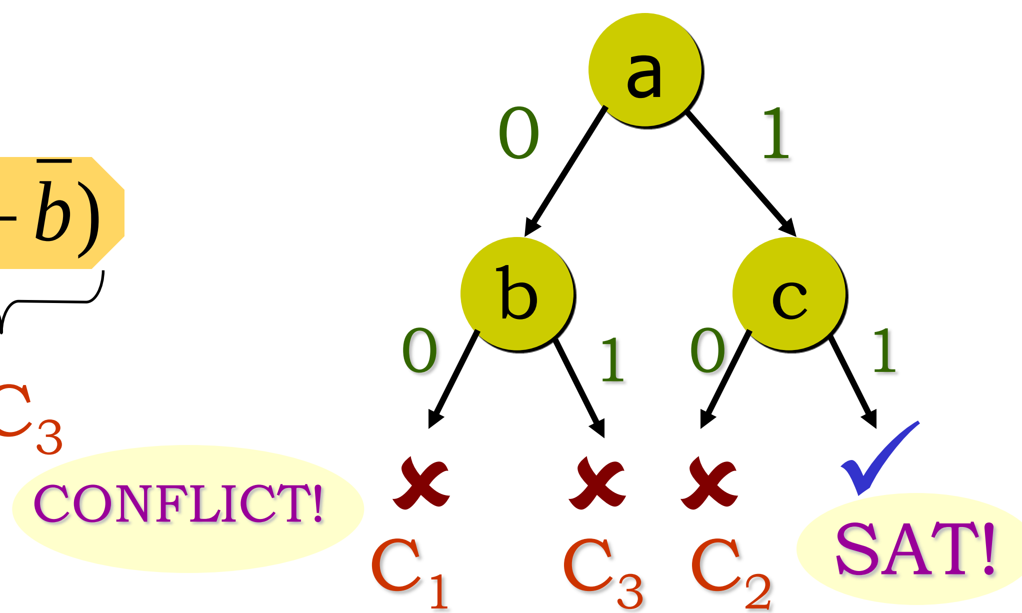
DPLL algorithm for SAT

[Davis, Putnam, Logemann, Loveland 1960,62]

Given : CNF formula $f(v_1,v_2,..,v_k)$, and an ordering function Next_Variable

Example :

$$\underbrace{(a+b)}_{C_1}\underbrace{(\bar{a}+c)}_{C_2}\underbrace{(a+\bar{b})}_{C_3}$$



DPLL algorithm: Unit clause rule

Rule: Assign to *true* any single literal clauses.

$$(a + b + c)$$

$a = 0$, $b = 0$ → $c = 1$

Apply Iteratively: *Boolean Constraint Propagation (BCP)*

$$a(\overline{a} + c)(\overline{b} + c)(a + b + \overline{c})(\overline{c} + e)(\overline{d} + e)(c + d + \overline{e})$$

$$\downarrow$$

$$c(\overline{b} + c)(\overline{c} + e)(\overline{d} + e)(c + d + \overline{e})$$

$$\downarrow$$

$$e(\overline{d} + e)$$

Anatomy of a modern SAT solver

DPLL Algorithm

Efficient BCP

Clause database management

- Discard *useless* clauses (*e.g.* inactive or large clauses)
- Efficient garbage collection

SAT Solver

Search Restarts

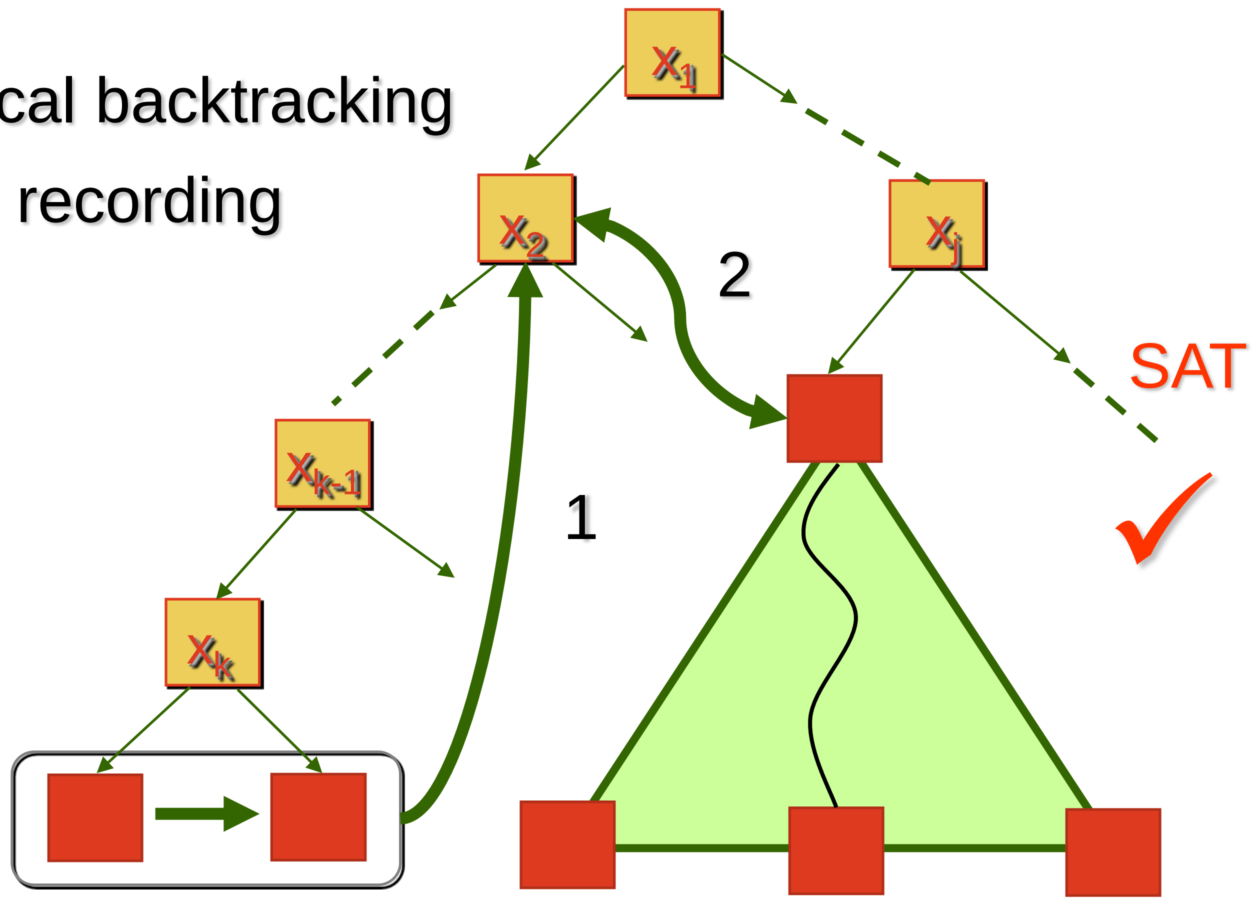
- To correct for bad choices in variable ordering
- Restart algorithm "periodically"
- Retain some/all recorded clauses

Conflict-driven learning

Conflict driven search pruning *(GRASP)*

Silva & Sakallah '95

1. Non-chronological backtracking
2. Conflict-clause recording



Variable ordering

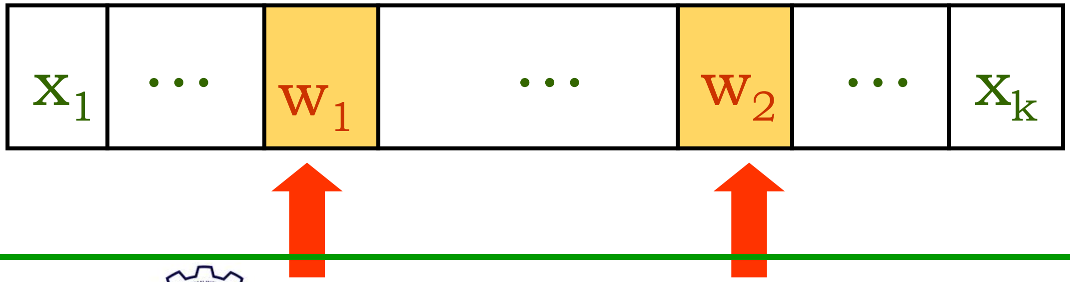
- Significantly impacts size of search tree
- Ordering schemes can be static or dymamic
- Conventional wisdom (pre-chaff):
  - Satisfy most number of clauses OR
  - Maximize BCP
  - *e.g.* DLIS, MOMs, BOHMs *etc.*

Variable ordering: New ideas

- New wisdom: Recorded clauses key in guiding search
- Conflict-driven variable ordering:
  - Chaff (DAC'01): Pick var. appearing in *most* number of *recent* conflict clauses
  - BerkMin (DATE'02): Pick var. *involved* in most number of *recent* conflicts
- Semi-static in nature, for efficiency
  - Statistics updated on each conflict
- Side-effect: Better cache behavior

Efficient Boolean Constraint Propagation

- Observation: BCP almost 80% of compute time, under clause recording
- Traditional implementation:
  - Each clause: Counter for #literals set to false
  - Assgn. to variable 'x': Update all clauses having x, $\overline{x}$
- New Idea: Only need to monitor event when # free literals in a clause goes from 2 to 1
  - Need to *watch* only 2 literals per clause : *SATO* (Zhang'97), *Chaff* (DAC'01)



SAT solvers today

- Capacity:
  - Formulas upto a *million variables* and *3-4 million clauses* can be solved in *few hours*
  - Only for *structured instances e.g.* derived from real-world circuits & systems

- Tool offerings:
  - Public domain
    - GRASP : Univ. of Michigan
    - SATO: Univ. of Iowa
    - zChaff: Princeton University
    - BerkMin: Cadence Berkeley Labs.
  - Commercial
    - PROVER: Prover Technologies

Thank you