# Canonical Representation

Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

*CS-226: Digital Systems*

*Lecture 8: 02 February 2021*

CADSL

# Canonical Forms

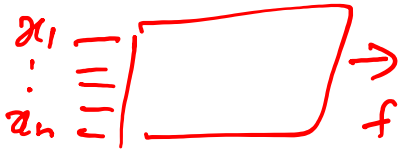
- Canonical Forms in common usage:
  - Truth Table → $2^n$
  - Sum of Minterms (SOM) $2^n$
  - Product of Maxterms (POM)
  - Binary Decision Diagram (BDD)
  - Reed Muller Representation

Textual.

graphical.
↑
compact

# Graphical Method

- BDD is canonical form of representation




- Shanon's expansion theorem
- $f(x_1, x_2, \ldots x_i, \ldots\ldots x_n) =$

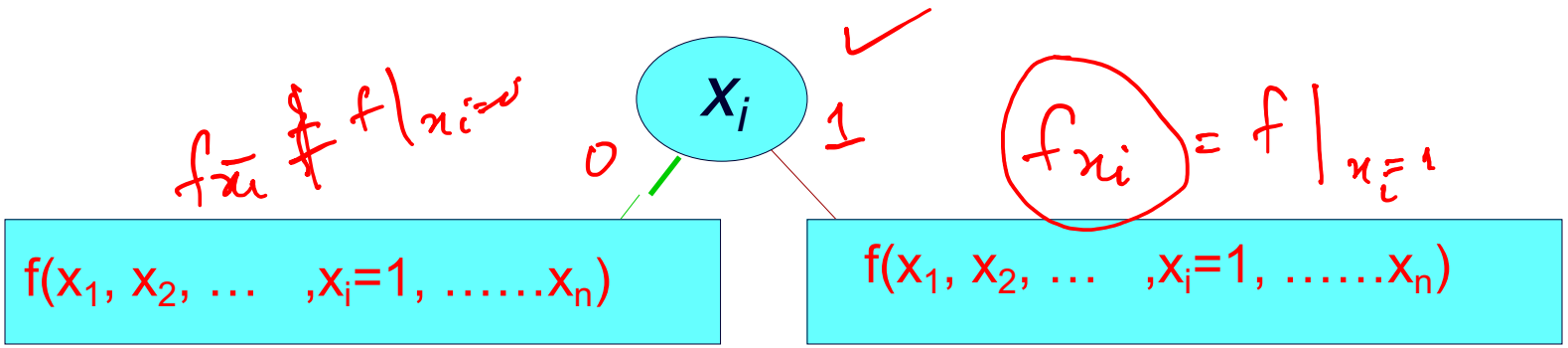
$$x_i.f(x_1, x_2, \ldots\ ,x_i=1, \ldots\ldots x_n) +$$

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

$$f(x_1, x_2, \ldots x_i, \ldots\ldots x_n) = x_i \cdot f_{x_i} + \bar{x}_i f_{\bar{x}_i}$$

$$= x_i\, f_{x_i} \oplus \bar{x}_i \cdot f_{\bar{x}_i}$$

$f|_{x_i=1}$ $\quad$ $f|_{x_i=0}$



# Decision Structures

$x_1 < x_2 < x_3$ order.

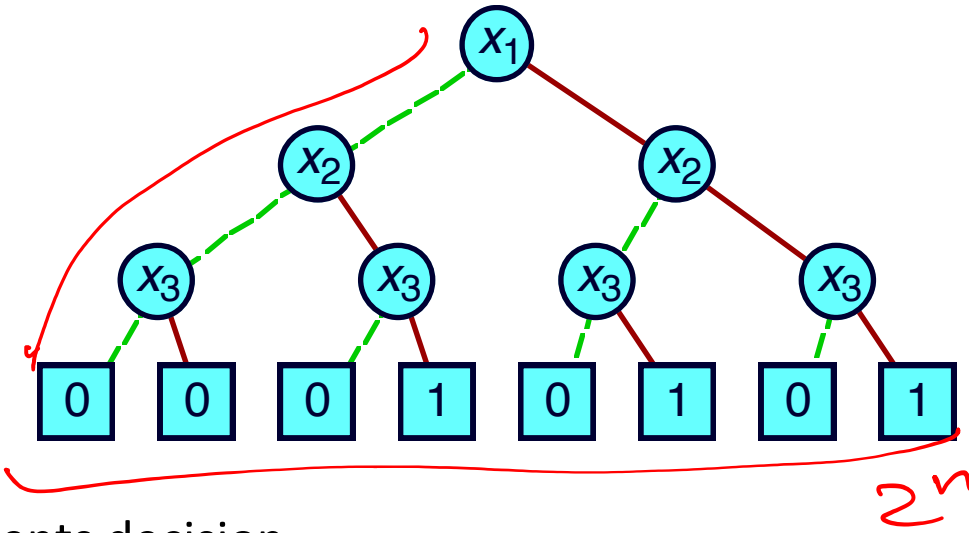
**Truth Table**

Binary **Decision Tree**

| $x_1$ | $x_2$ | $x_3$ | $f$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

$2^n$

$2^{n+1} - 1$

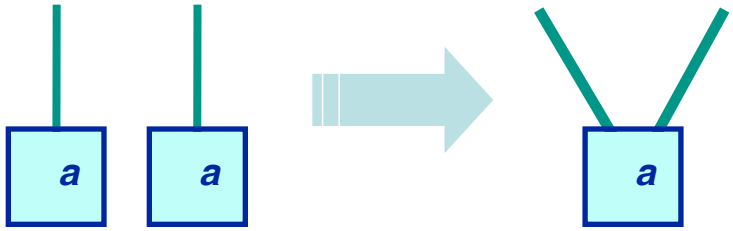


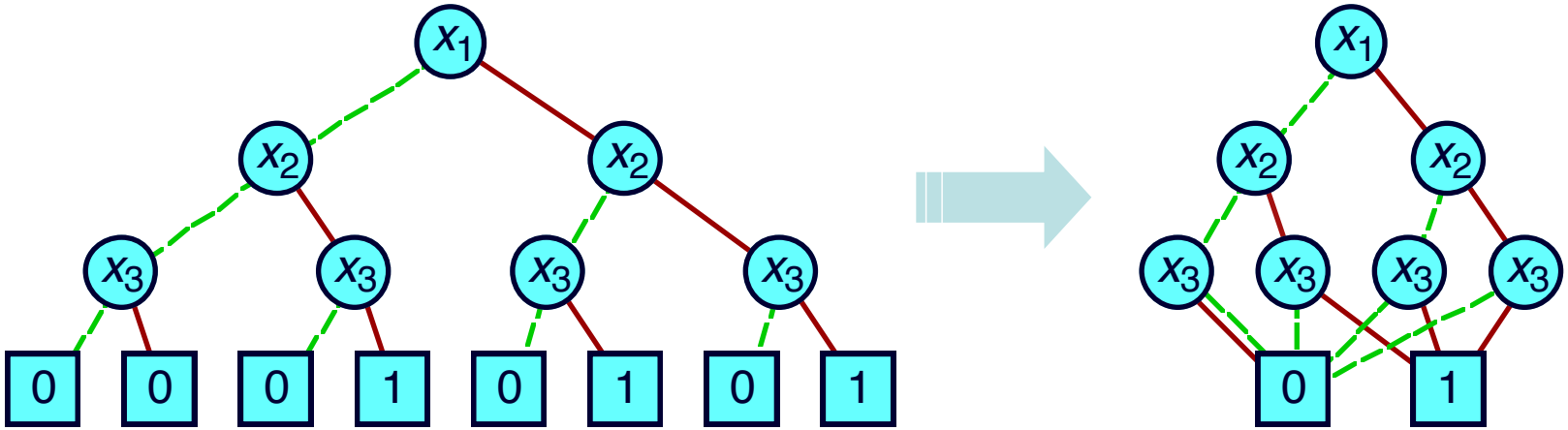

$2^n$

- Vertex represents decision
- Follow green (dashed) line for value 0
- Follow red (solid) line for value 1
- Function value determined by leaf value.

# Reduction Rule #1

**Merge equivalent leaves**

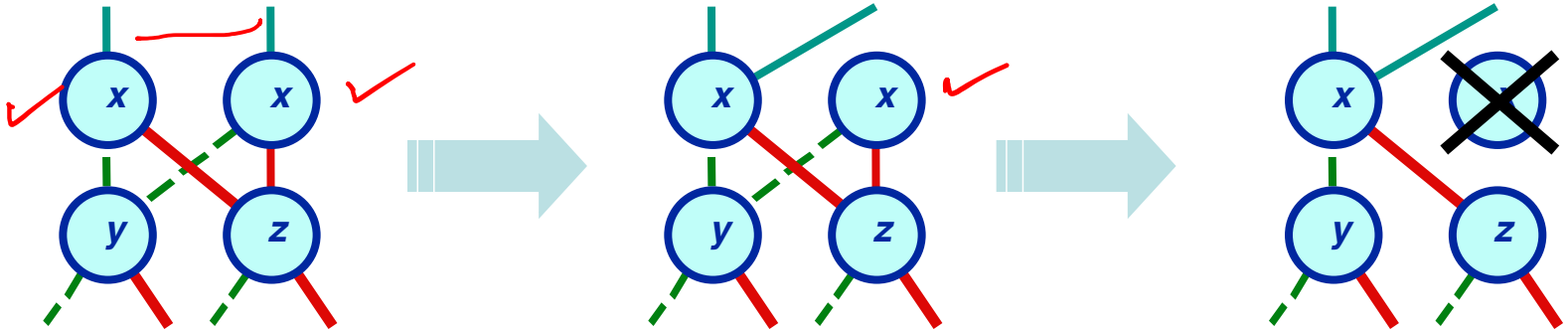


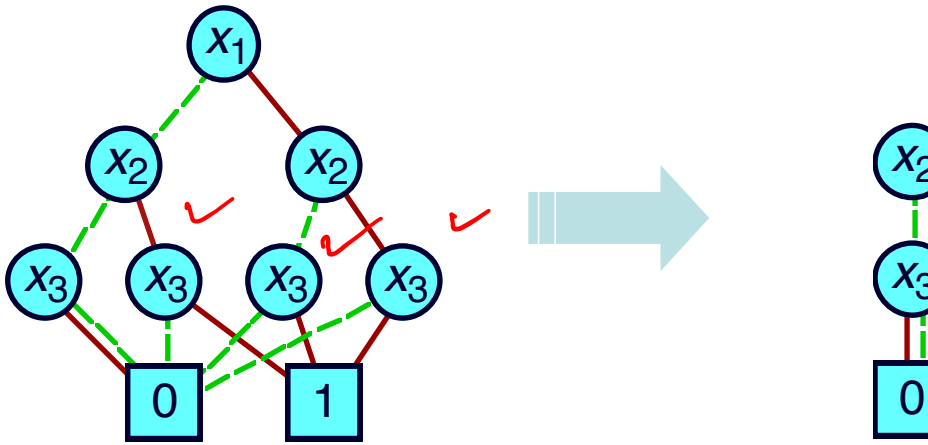




$2^n + 1$

# Reduction Rule #2

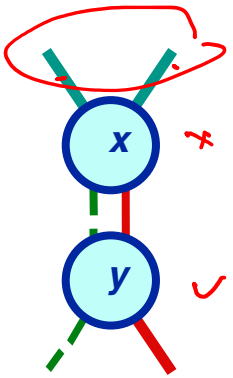
**Merge isomorphic nodes**

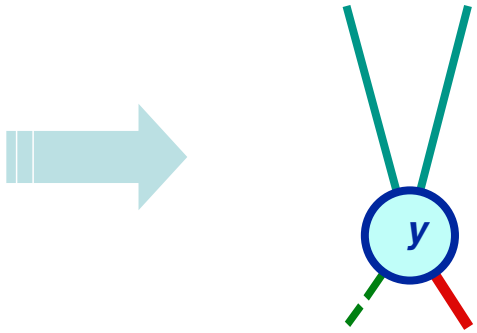


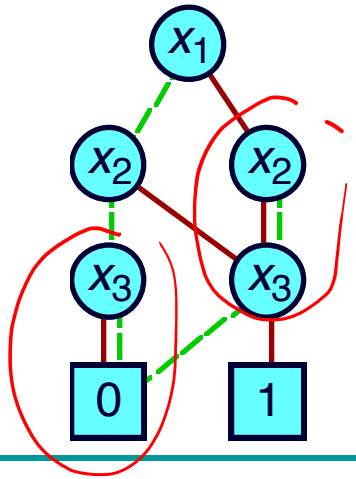



# Reduction Rule #3

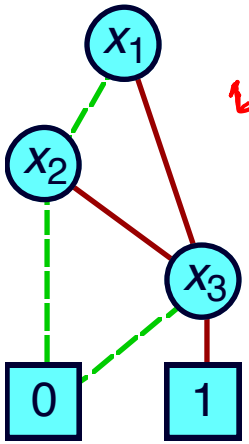
**Eliminate Redundant Tests**



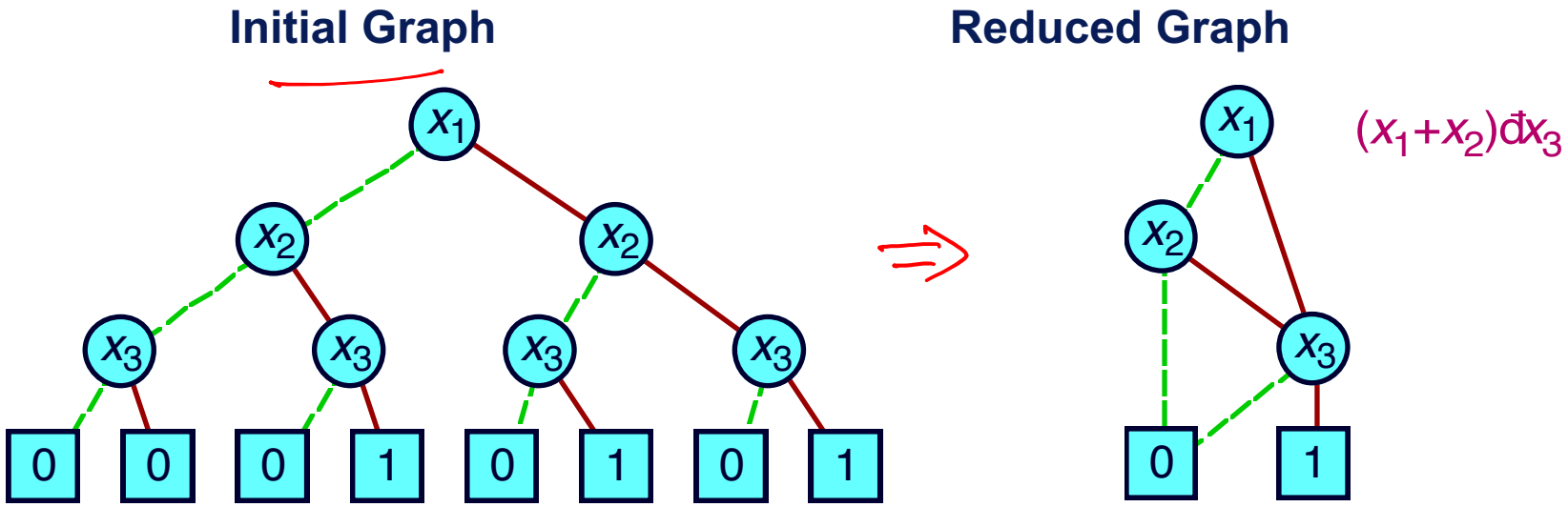





# Example OBDD

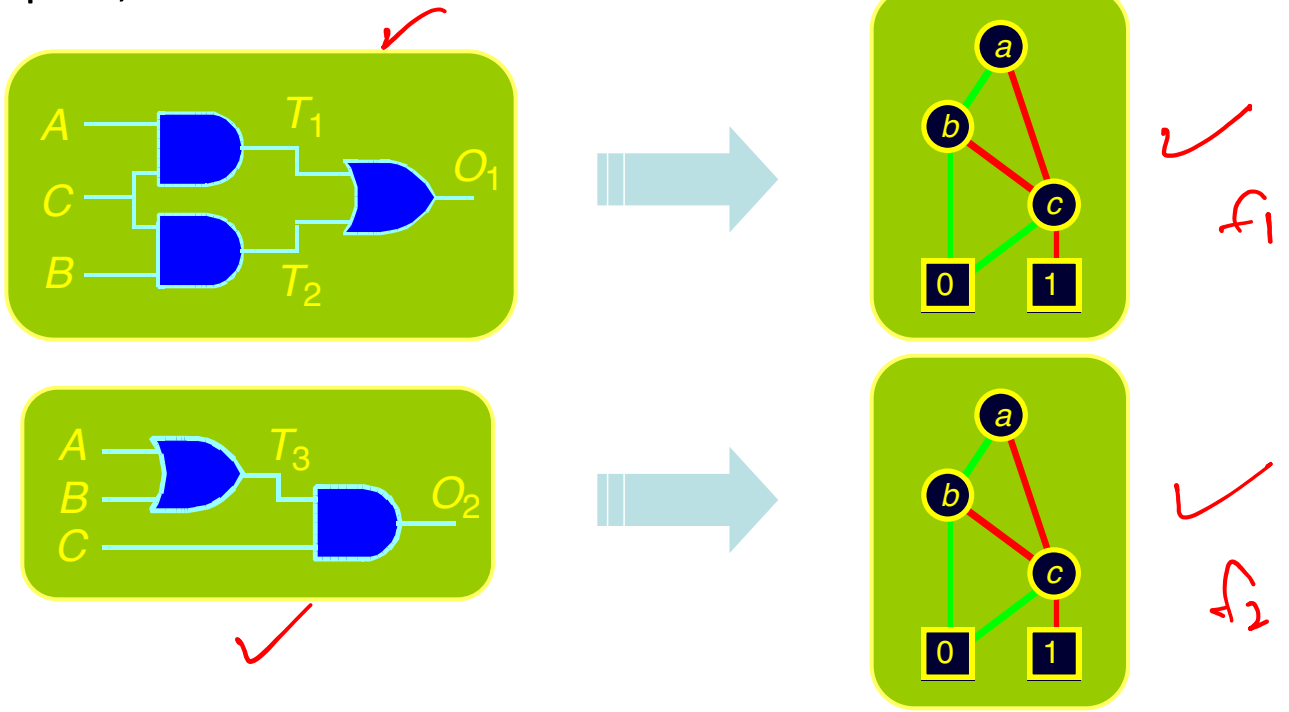



- Canonical representation of Boolean function
  - ❖ For given variable ordering
  - ➢ Two functions equivalent if and only if graphs isomorphic
    - o Can be tested in linear time
  - ➢ Desirable property: *simplest form is canonical*.

# Binary Decision Diagram

ROBDD

- Generate Complete Representation of Circuit Function
  - Compact, canonical form




- Functions equal if and only if representations identical
- Never enumerate explicit function values
- Exploit structure & regularity of circuit functions

$f_1 \equiv f_2$

Isomorphic

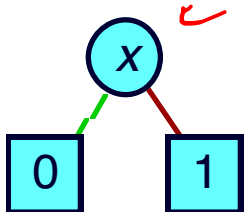
# Example Functions

ROBDD

## Constants

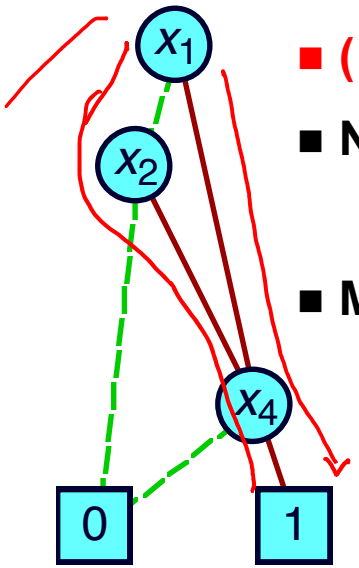
- 0 — Unique unsatisfiable function
- 1 — Unique tautology

## Variable




Treat variable as function

## Typical Function

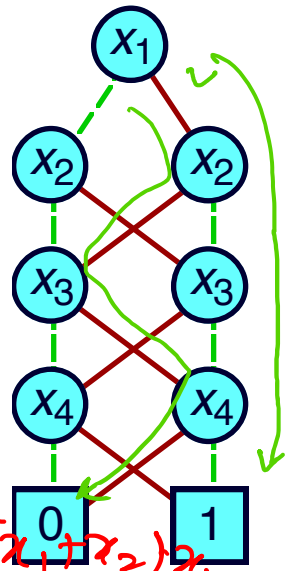



- $( x_1 + x_2 ) . x_4$
- **No vertex labeled $x_3$**
  - independent of $x_3$
- **Many subgraphs shared**

$x_1 \cdot x_4 + \bar{x}_1 \cdot x_2 x_4$

$= (x_1 + \bar{x}_1 x_2) x_4 = (x_1 + x_2) x_4$

## Odd Parity




$f(x_1, x_2, x_3, x_4)$

**Linear representation**

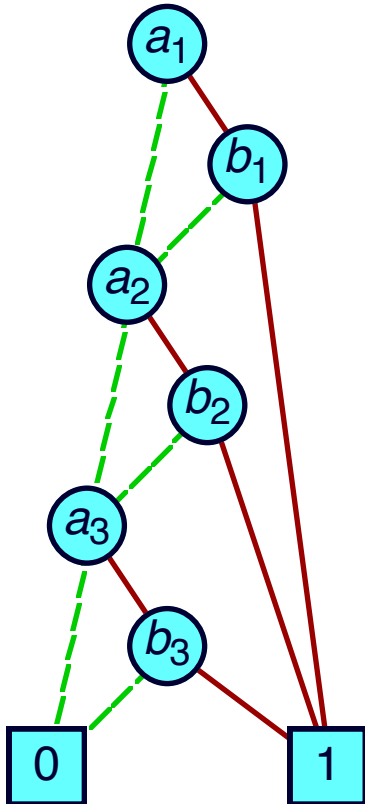
# Effect of Variable Ordering

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

$a_1 < b_1 < a_2 < b_2 < a_3 < b_3$

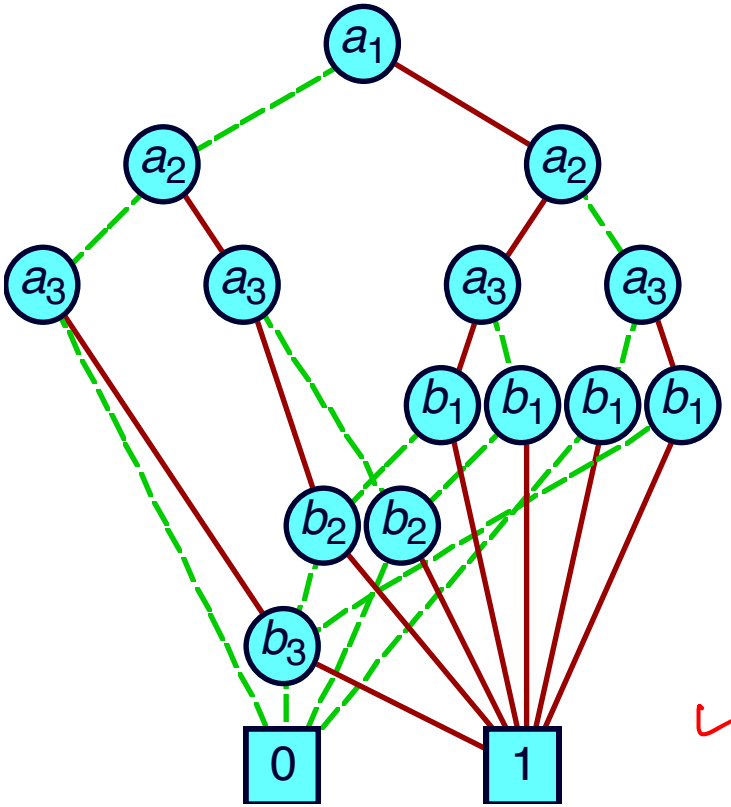
$a_1 < a_2 < a_3 < b_1 < b_2 < b_3$

**Good Ordering**




**Linear Growth**

**Bad Ordering**




**Exponential Growth**

# Selecting Good Variable Ordering

- Intractable Problem
  - Even when problem represented as OBDD
    - i.e., to find optimum improvement to current ordering
- Application-Based Heuristics
  - Exploit characteristics of application
  - e.g., Ordering for functions of combinational circuit
    - Traverse circuit graph depth-first from outputs to inputs
    - Assign variables to primary inputs in order encountered

# Selecting Good Variable Ordering

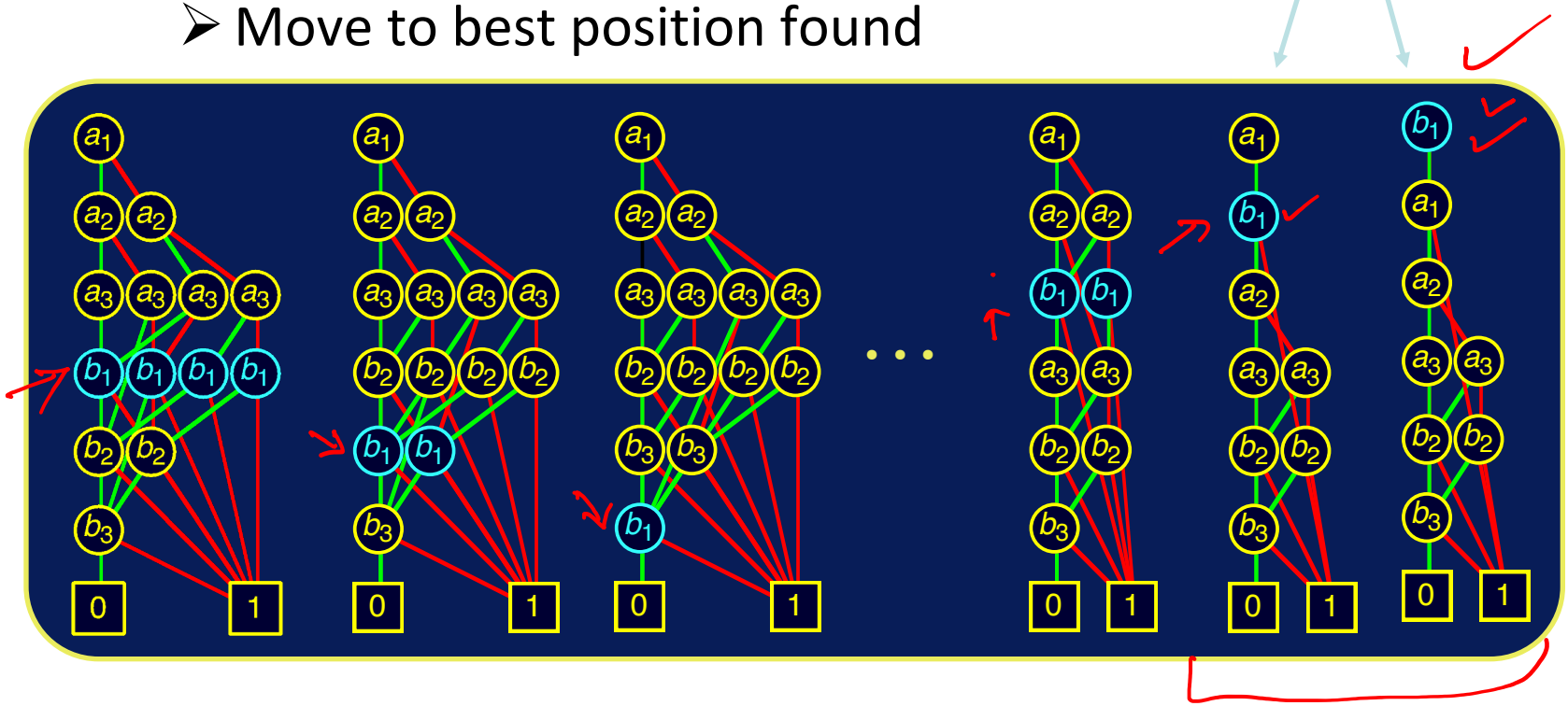
- Static Ordering
  - Fan In Heuristic
  - Weight Heuristic
- Dynamic Ordering
  - Variable Swap
  - Window Permutation
  - Sifting

# Dynamic Variable Reordering

- Richard Rudell, Synopsys
- Periodically Attempt to Improve Ordering for All BDDs
  - ❖ Move each variable through ordering to find its best location
- Has Proved Very Successful
  - ❖ Time consuming but effective

# Dynamic Reordering By Sifting

- Choose candidate variable
- Try all positions in variable ordering
  - Repeatedly swap with adjacent variable
- Move to best position found

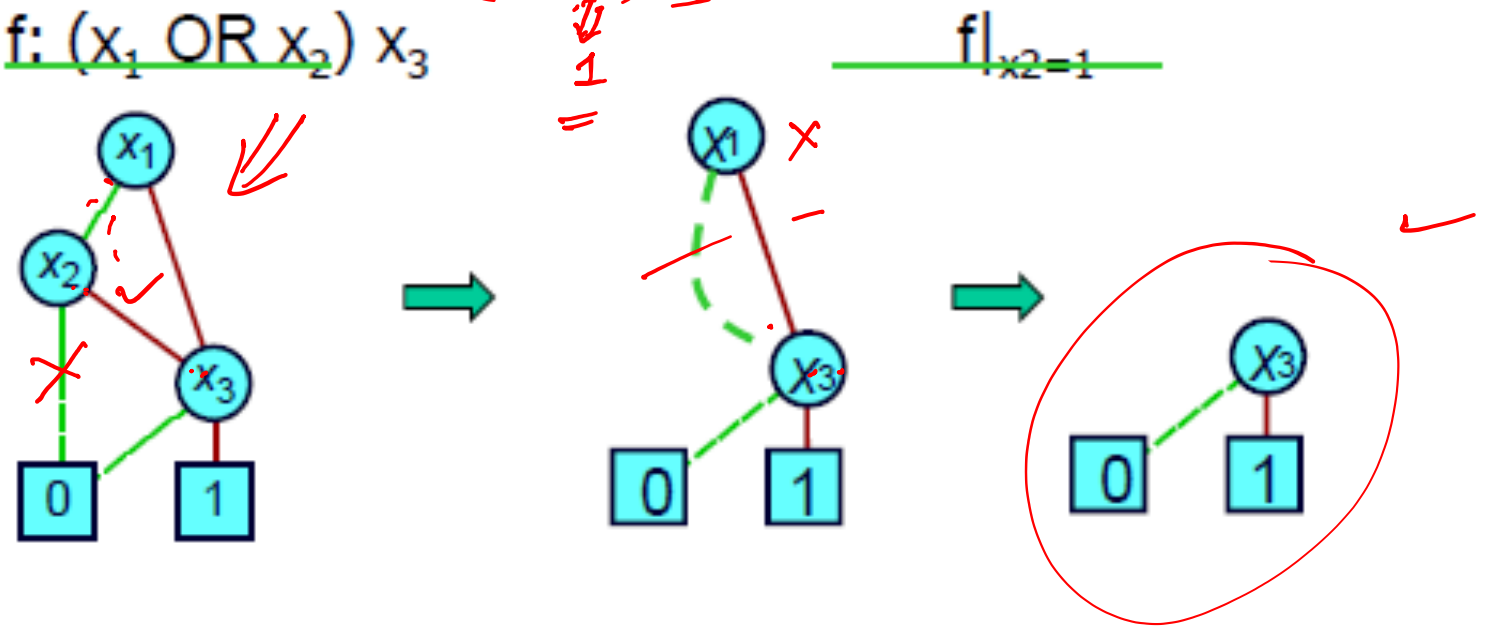


# ROBDD Sizes & Variable Ordering

- 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 1,000,000 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

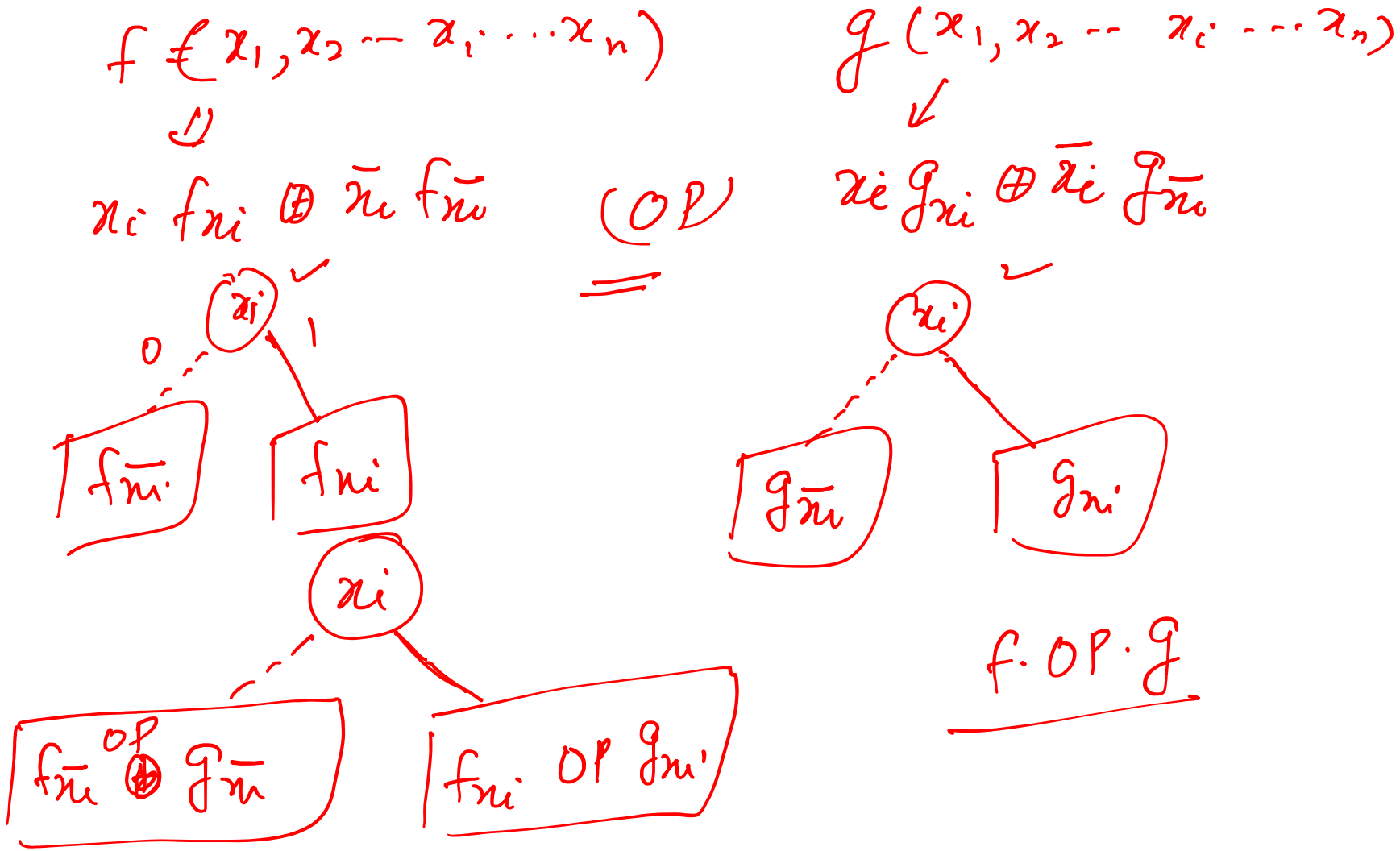
- Restriction: A restriction to a function to x=d, denoted $f|_{x=d}$, where x ϵ var (f), and d ϵ {0,1}, is equal to f after assigning x = d.
- Given BDD of f, deriving BDD of $f|_{x=d}$ is simple



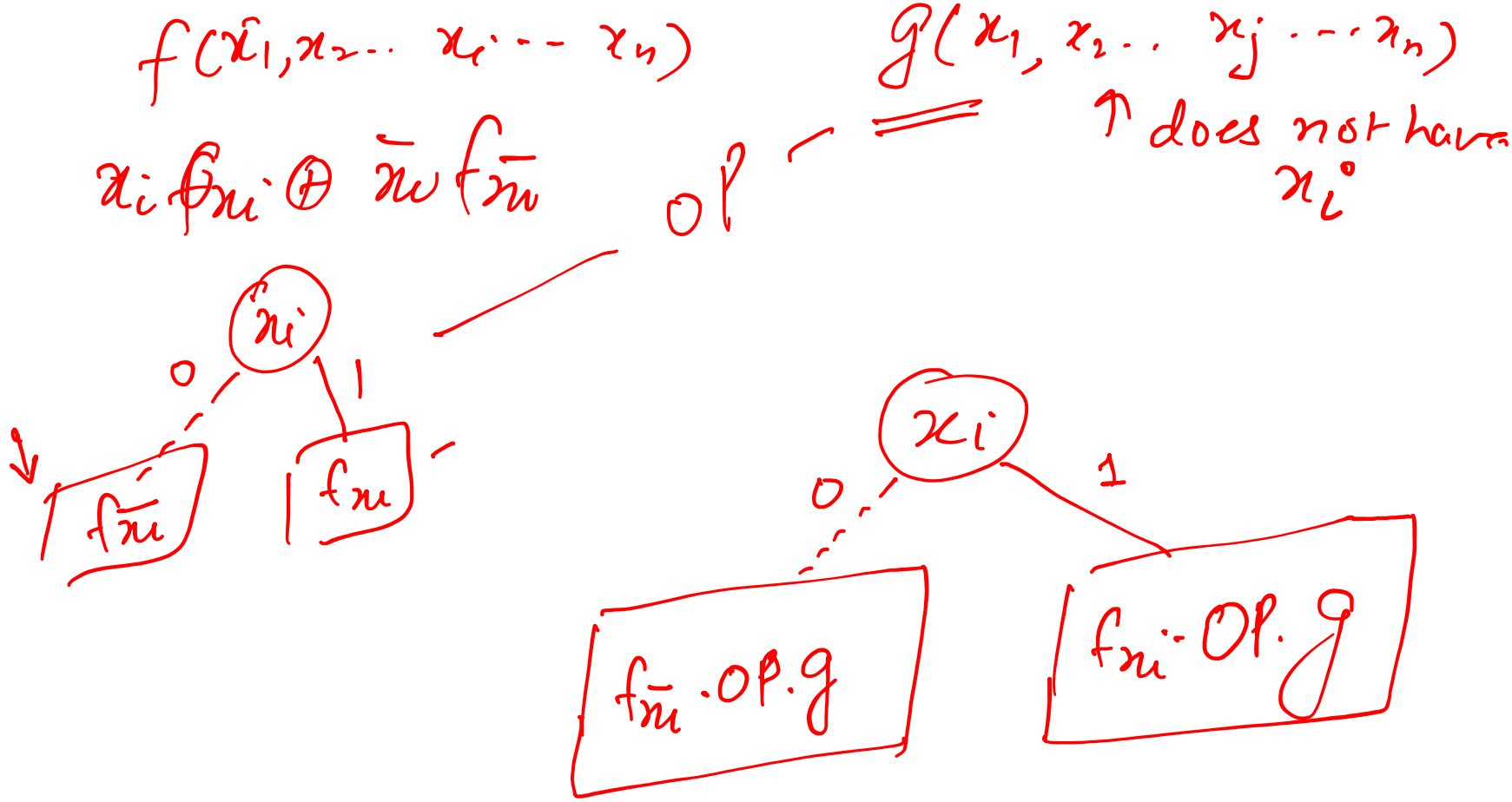
# Operations with BDD

- 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



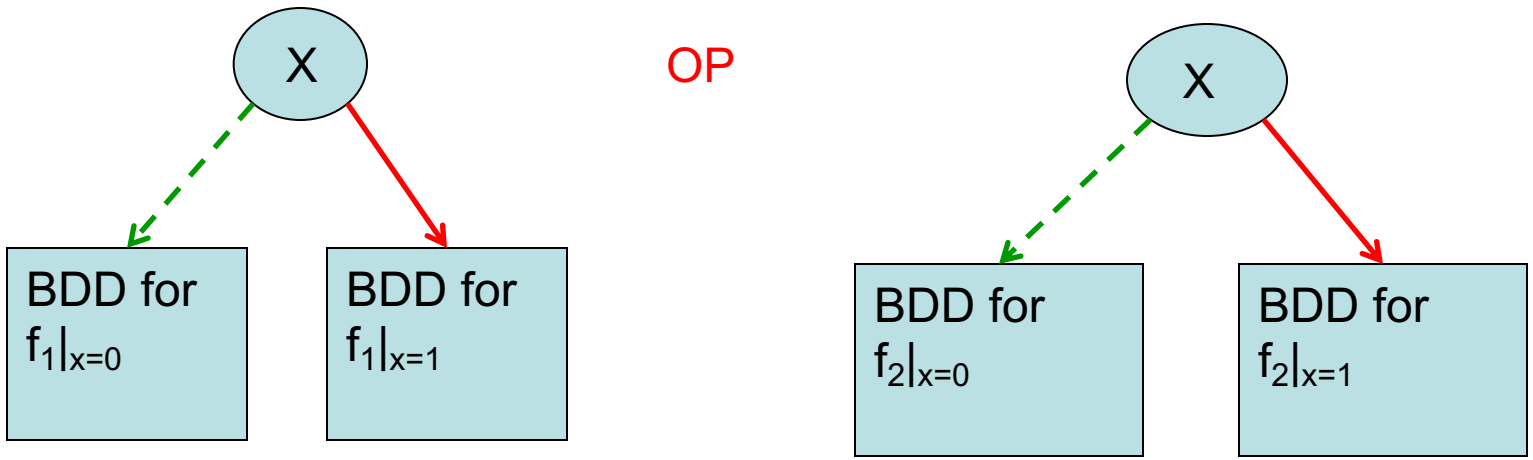
# Operations with BDD



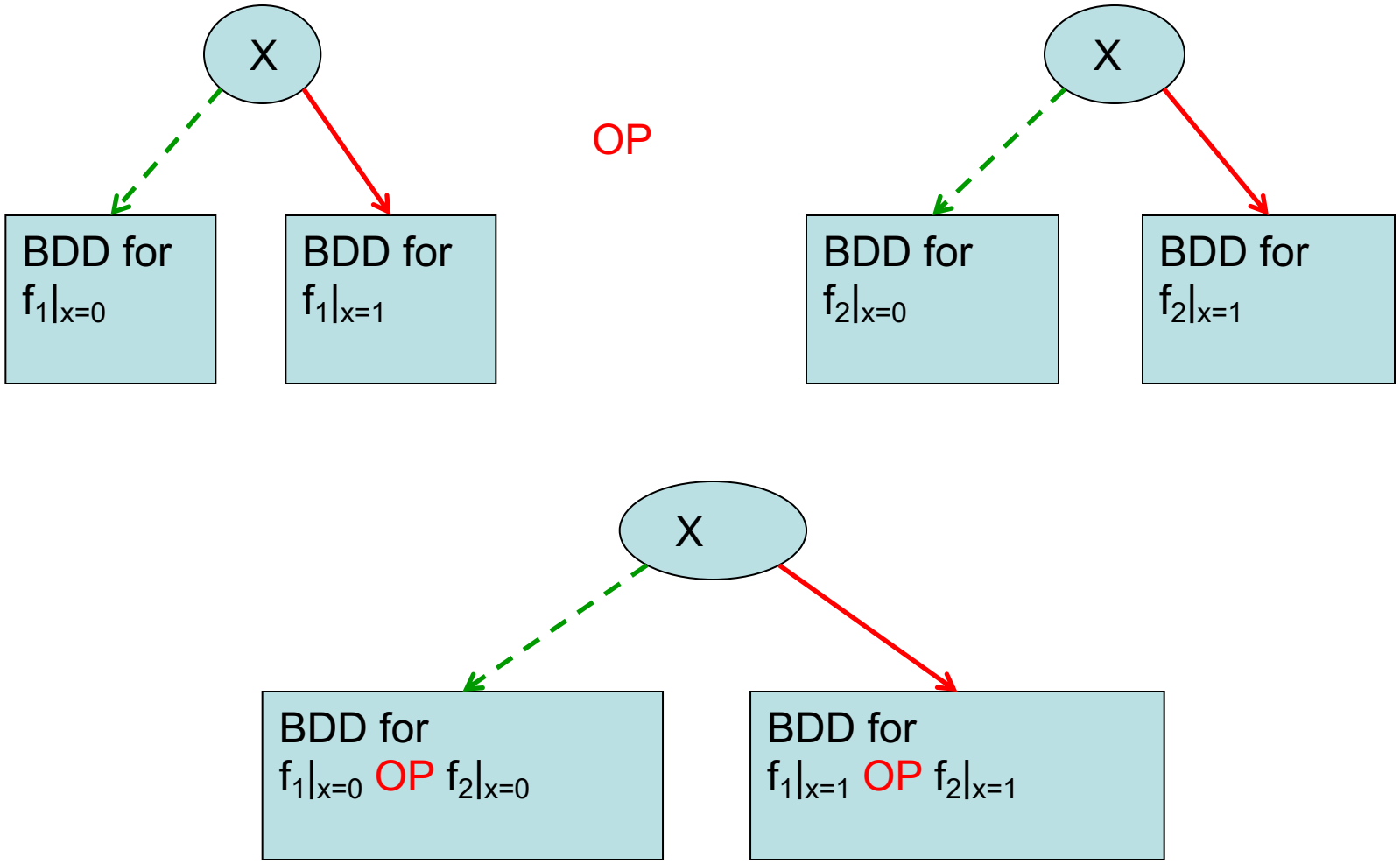
# Operations with BDD

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

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



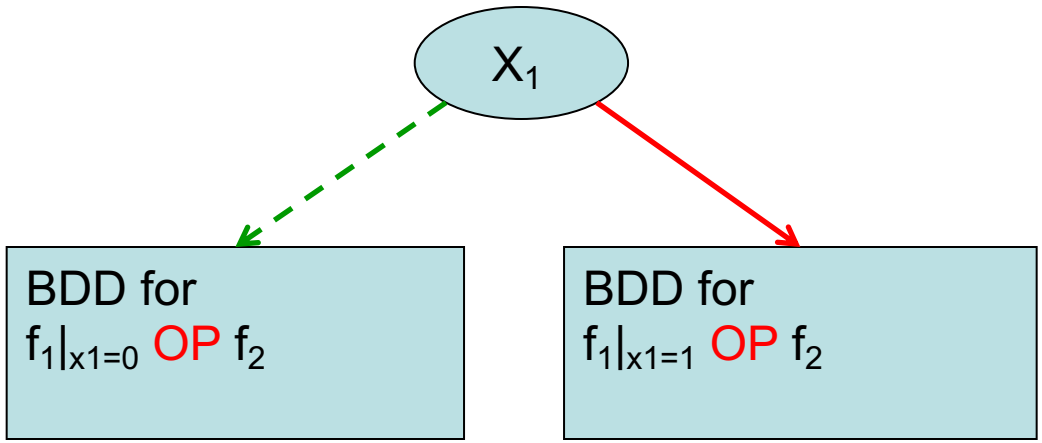
# Operations with BDD



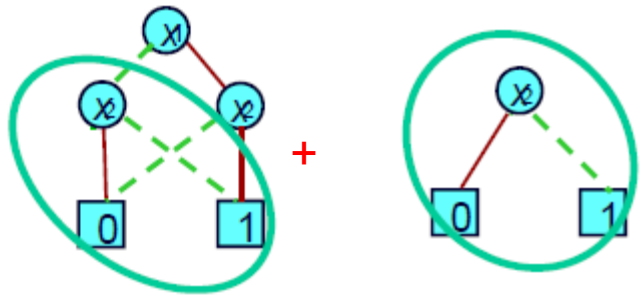
# Operations with BDD

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

$$f_1 \text{ OP } f_2 = x'_1 (f_1|_{x1=0} \text{ OP } f_2) + x_1 (f_1|_{x1=1} \text{ OP } f_2)$$

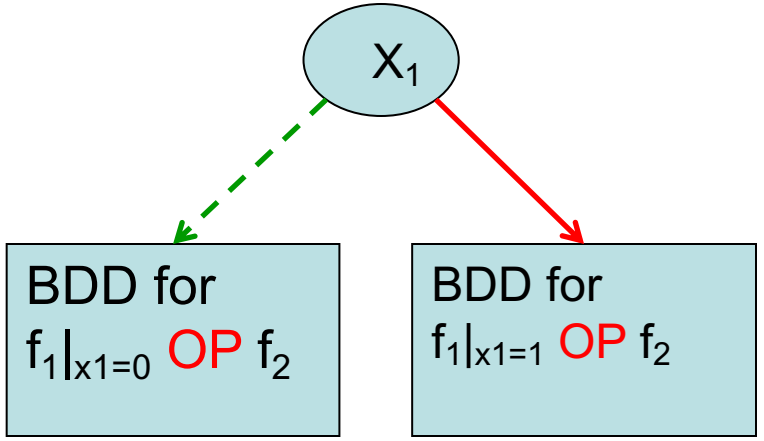


# Operations with BDD: Example

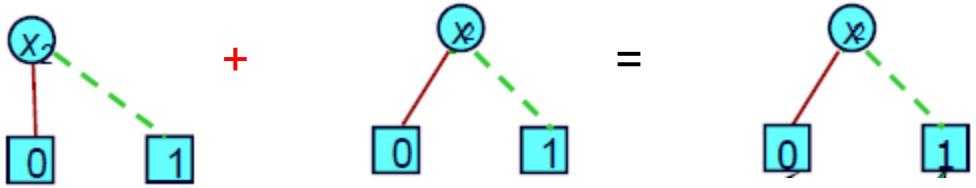



$f_1 = X_1 \text{ XNOR } X_2$

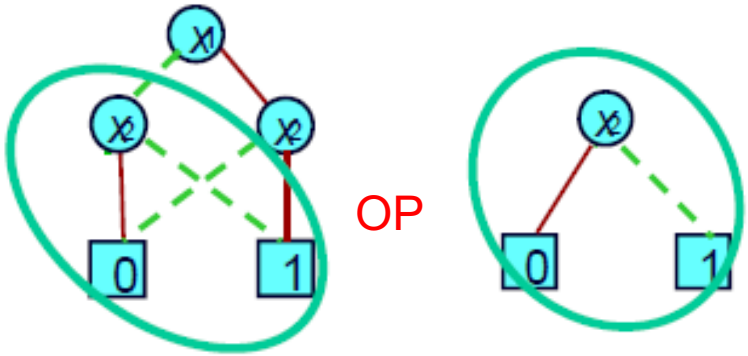
$f_2 = X'_2$




BDD for $f_1|_{x1=0}$ OP $f_2$ =



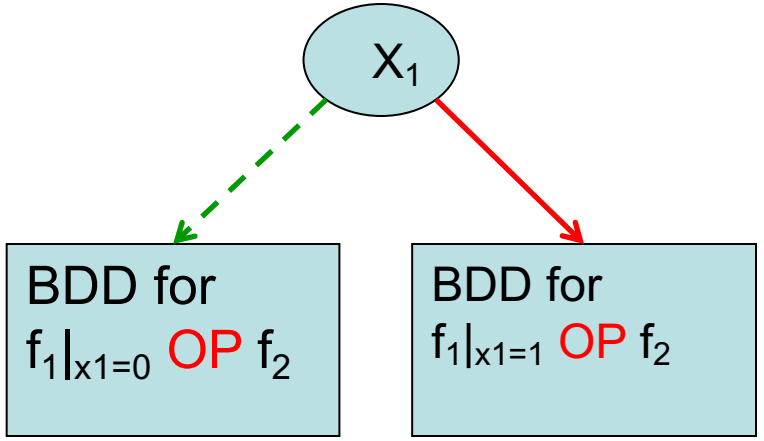
# Operations with BDD: Example




OP



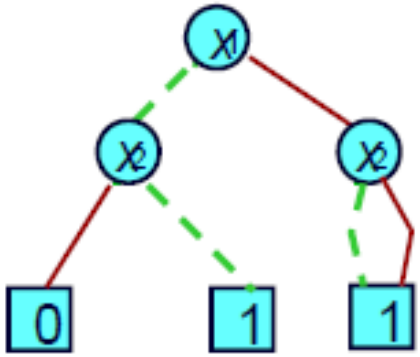

=



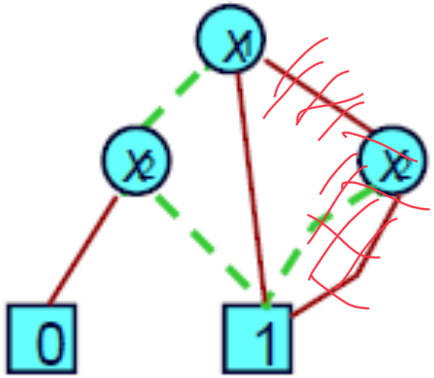

$f_1 = X_1 \text{ XNOR } X_2$

$f_2 = X'_2$

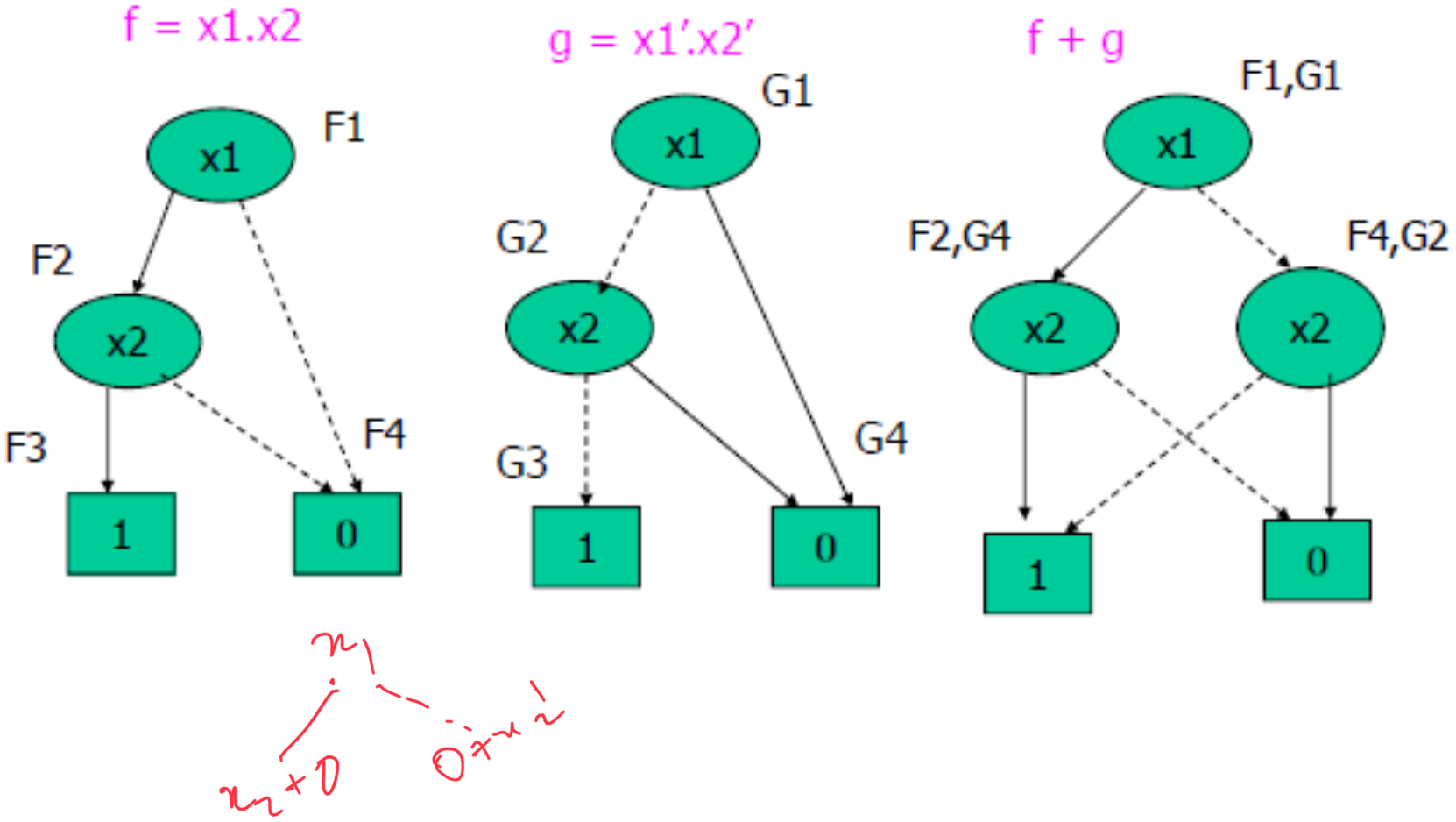
=



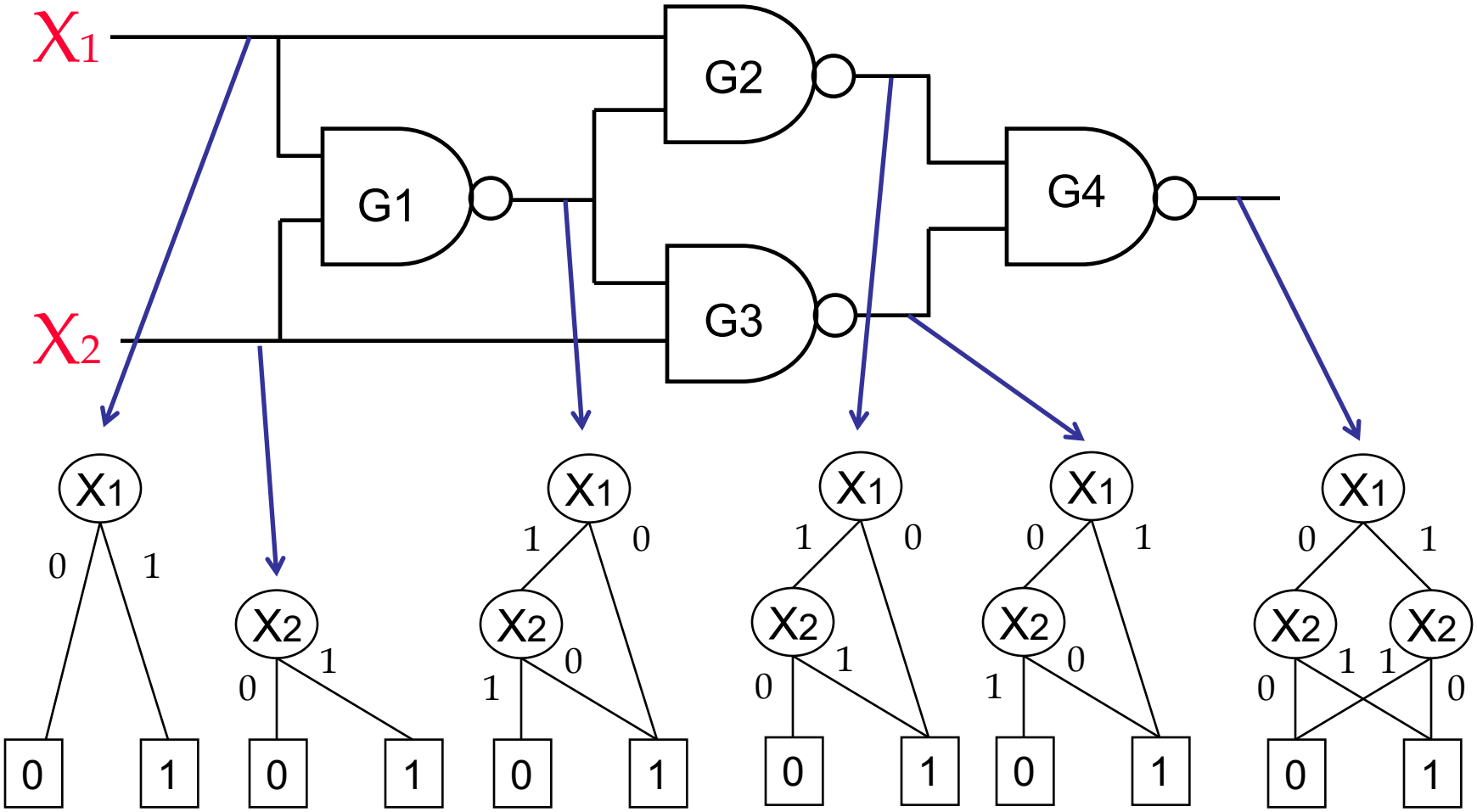

=



# Operations with BDD: Example



# From Circuits to BDD



# Thank You