Formal Verification – 3
ROBDD

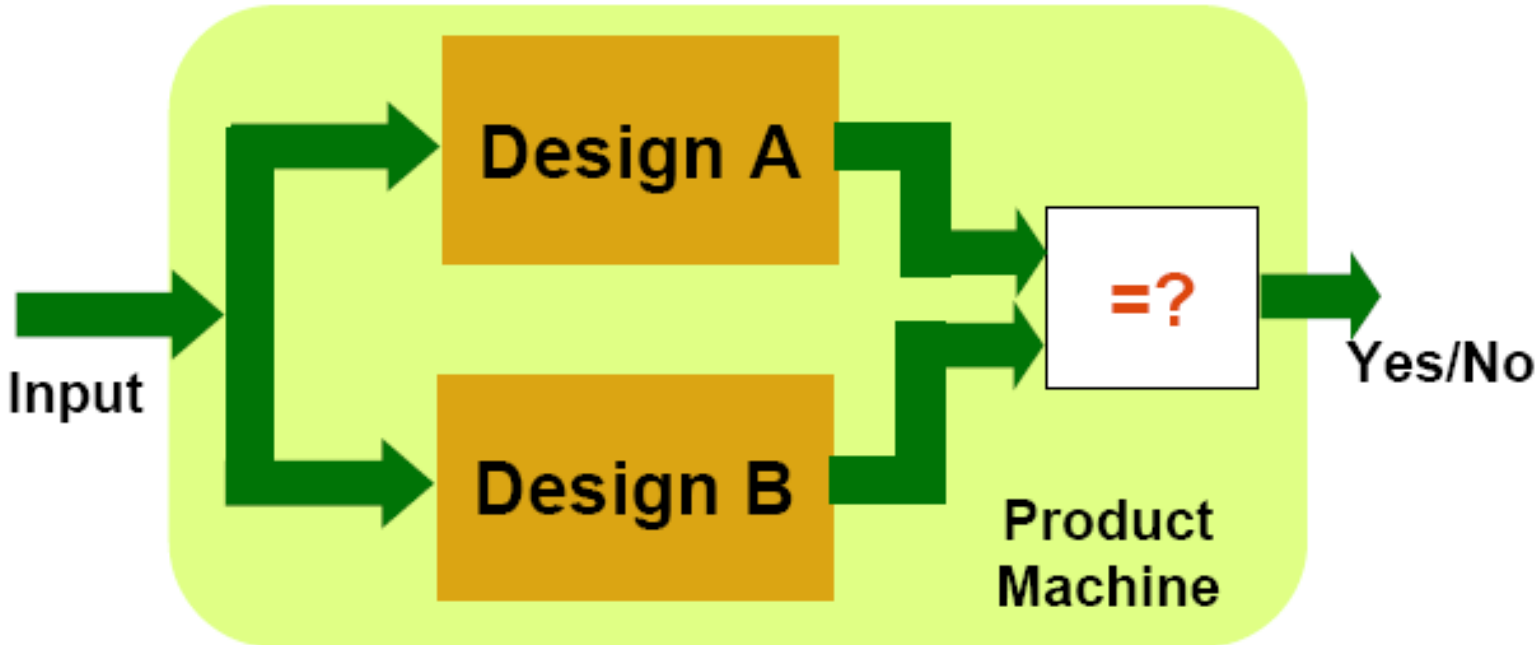
Virendra Singh
IC Design Group
CEERI, Pilani

MEL G 626: VLSI Test & Testability

Lecture - 11

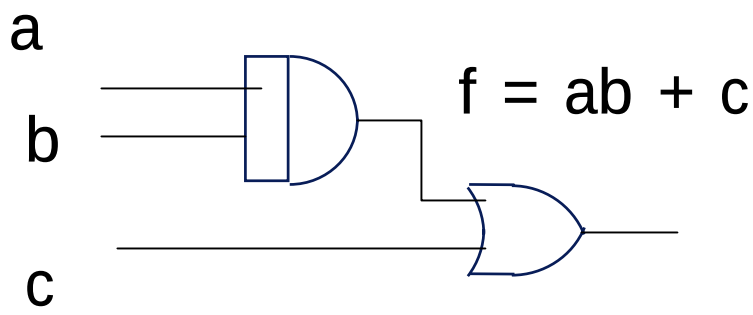
Formal Equivalence Checking

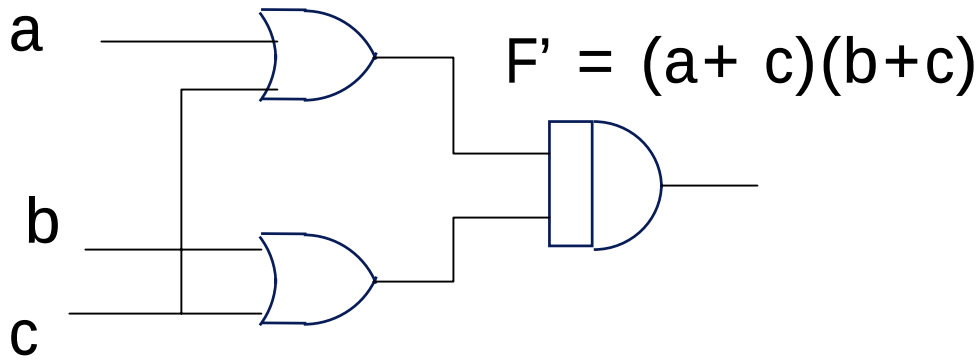
Given two designs, prove that for all possible input stimuli their corresponding outputs are equivalent



Formal Equivalence Checking

Canonical Forms

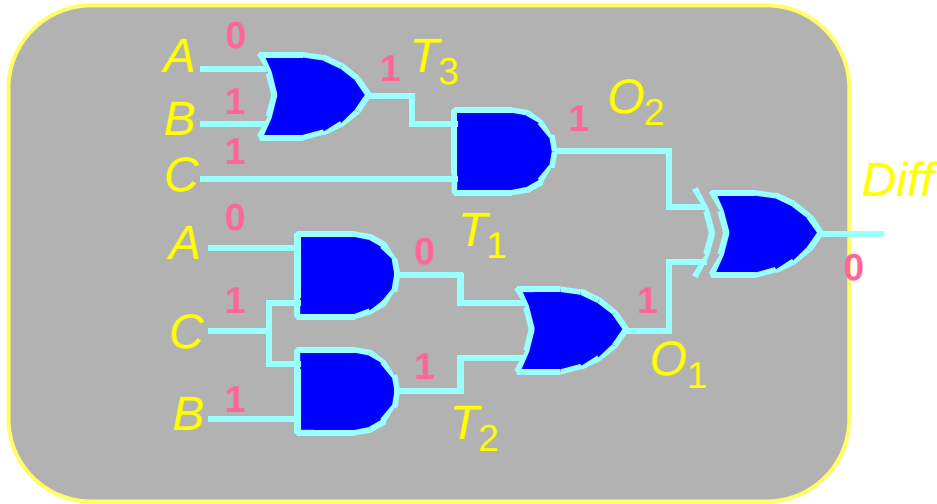




a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Formal Equivalence Checking

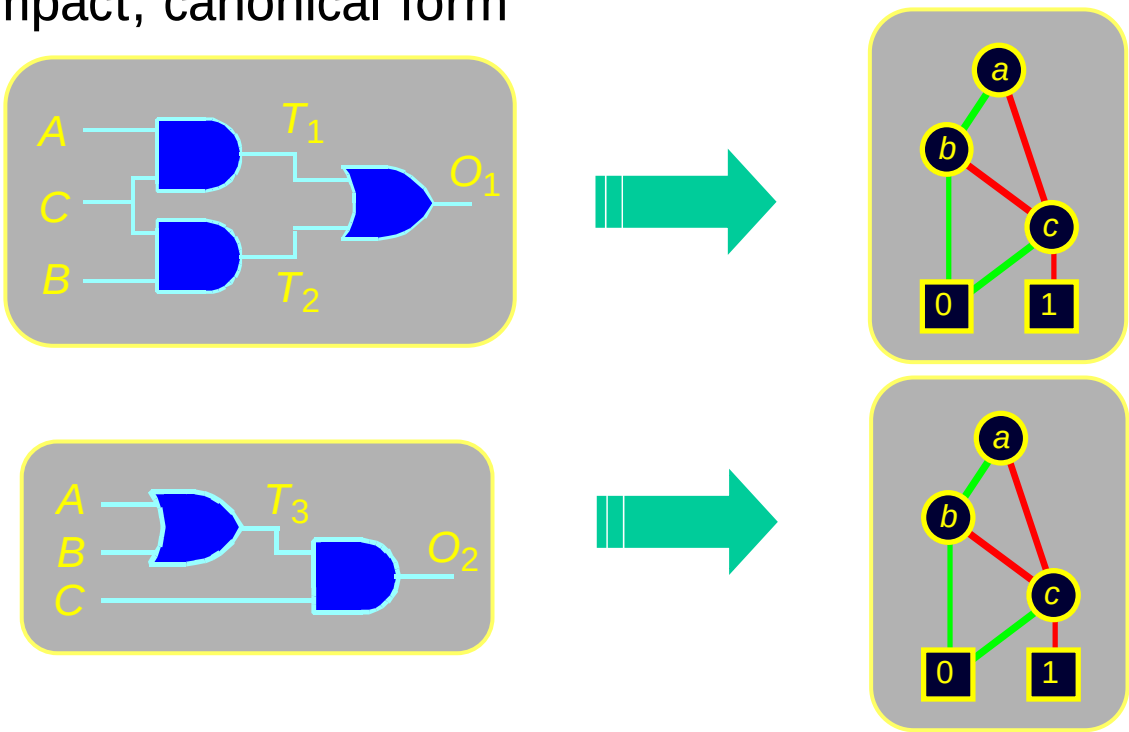
- Satisfiability Formulation
  - Search for input assignment giving different outputs
- Branch & Bound
  - Assign input(s)
  - Propagate forced values
  - Backtrack when cannot succeed



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

Binary Decision Diagram

- 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

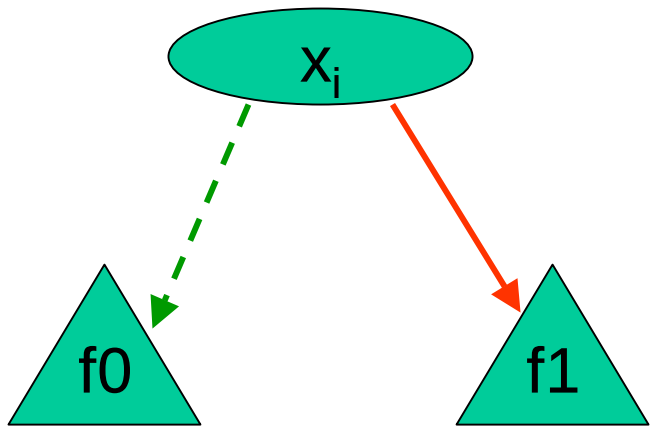
Binary Decision Diagram

Properties

- Reduced OBDDs provide a canonical represenation of a circuit
- Reduced OBDD can be manipulated efficiently
- For many practically important switching functions, the corresponding OBDD representations are quite small

Binary Decision Diagram

- A node v with label x_i defines a Shanon expansion
- If the OBDD rooted in v represents the function $f(x_1, x_2, \ldots, x_n)$ the
  - Two sub-OBDDs rooted in the sons represent the functions $f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$, and $f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$,

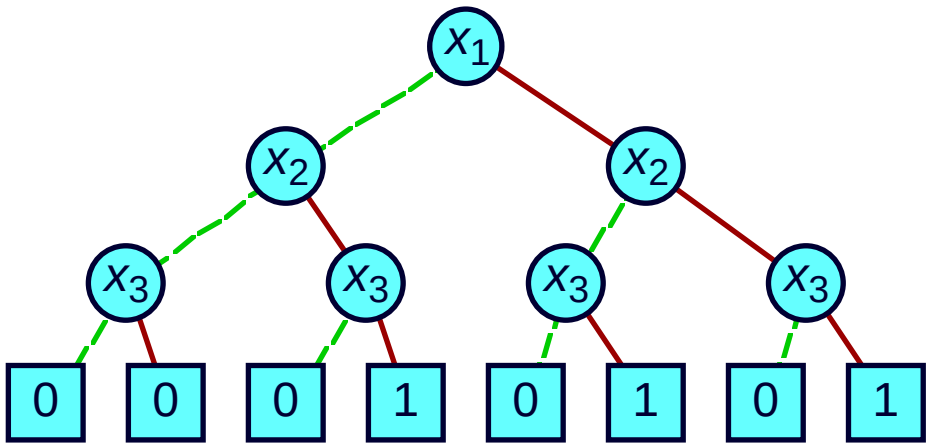


Decision Structures

Truth Table

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

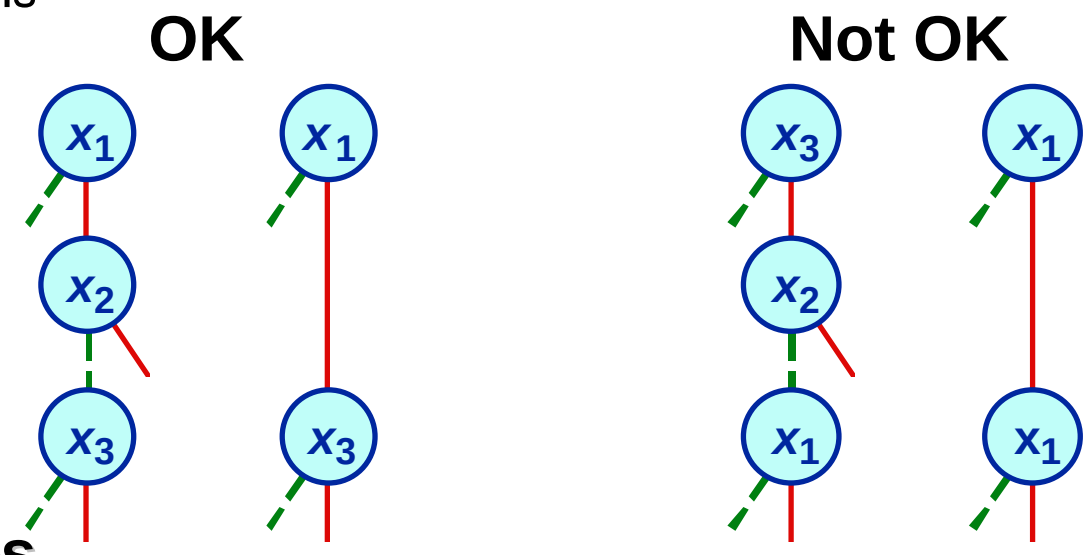
Decision Tree



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

Variable Ordering

- Assign arbitrary total ordering to variables
  - e.g., $x_1 < x_2 < x_3$
- Variables must appear in ascending order along all paths

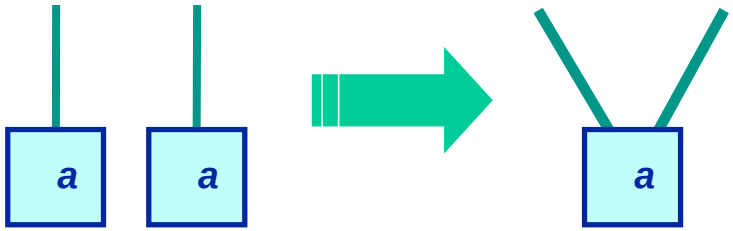


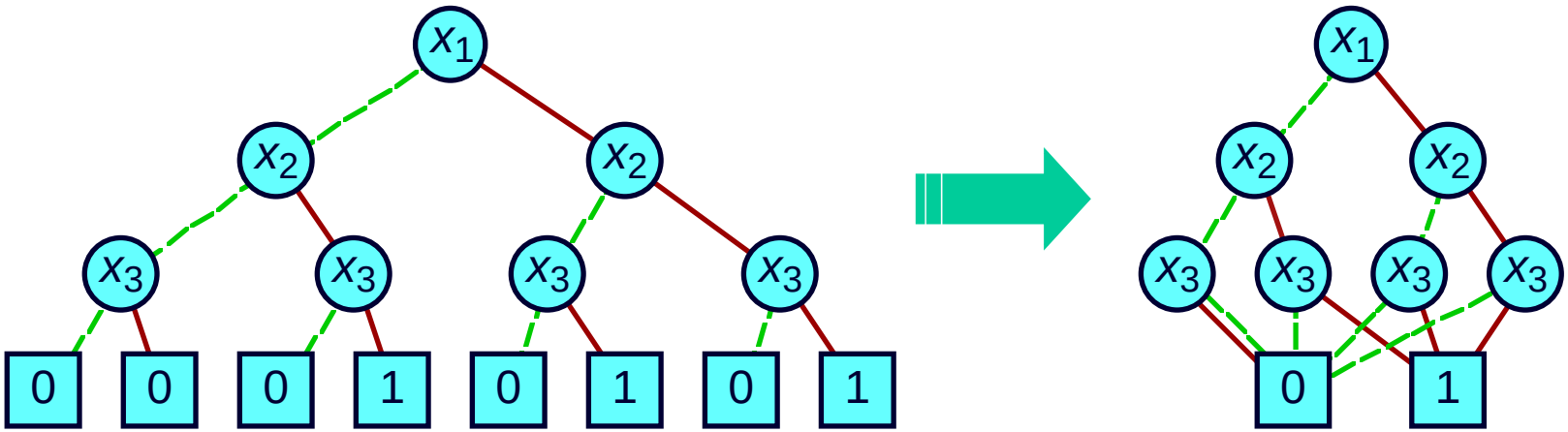
Properties

- No conflicting variable assignments along path
- Simplifies manipulation

Reduction Rule #1

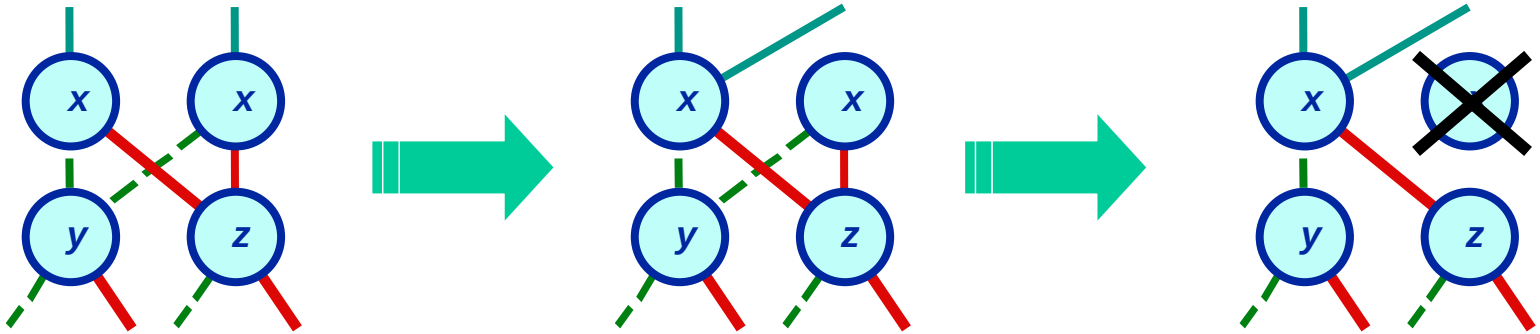
Merge equivalent leaves

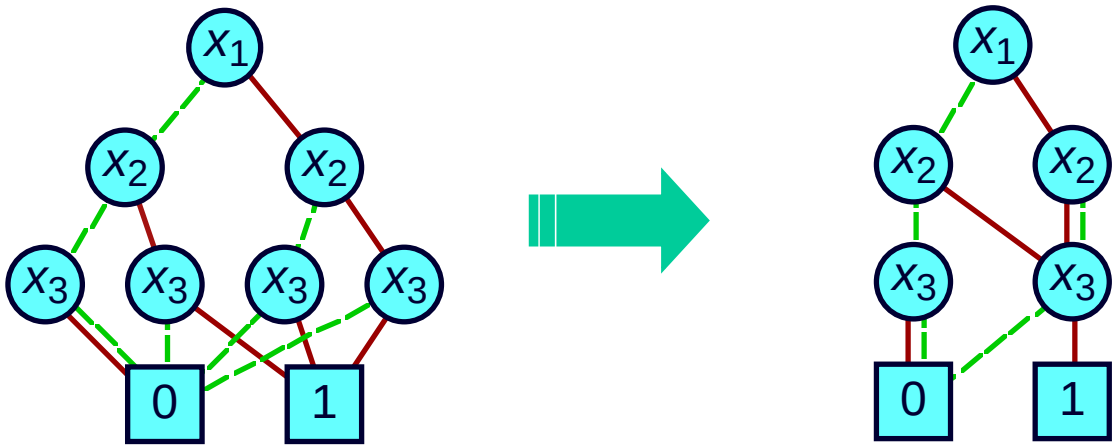




Reduction Rule #2

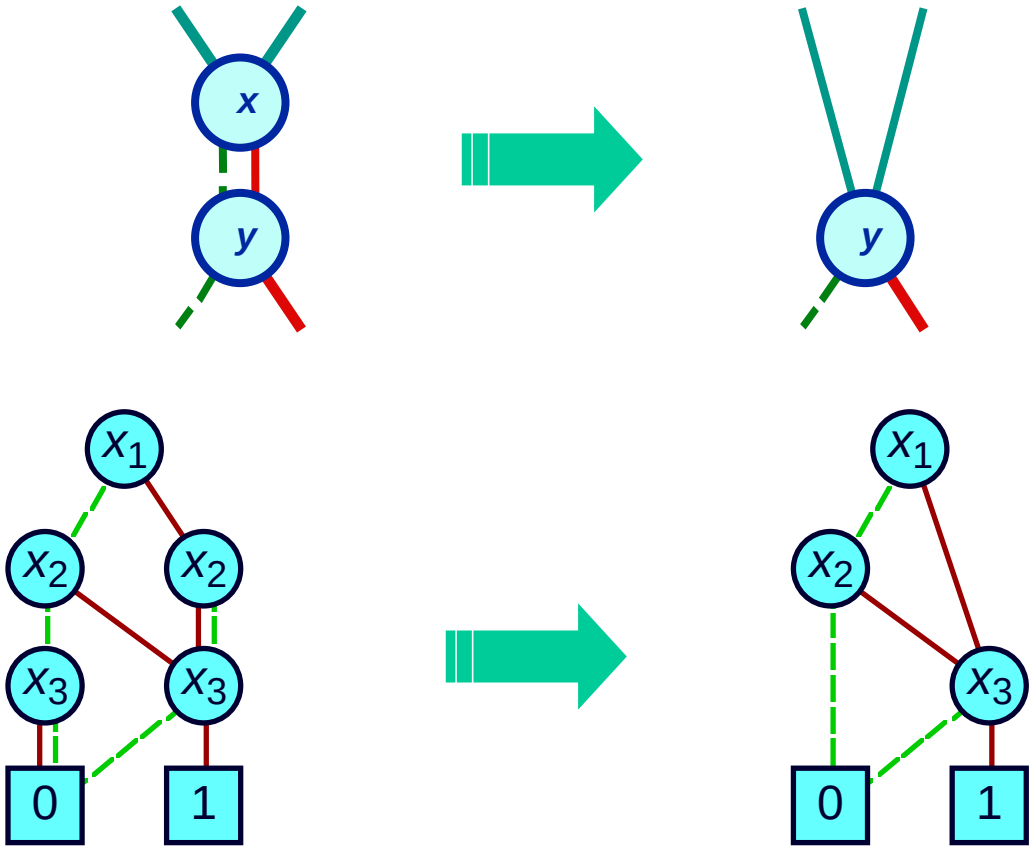
Merge isomorphic nodes





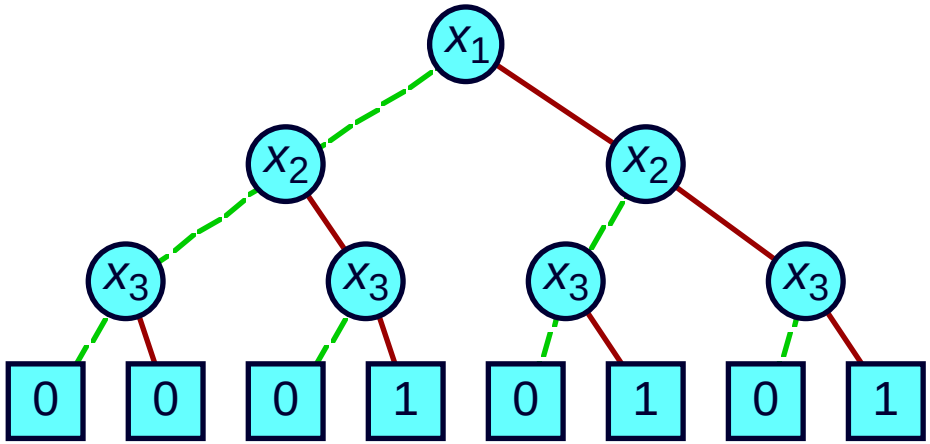
Reduction Rule #3

Eliminate Redundant Tests

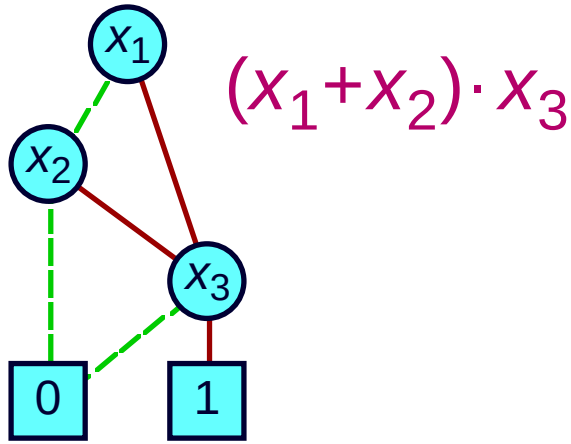


Example OBDD

Initial Graph



Reduced Graph



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

Reduced OBDD

Def: An OBDD is called reduced if

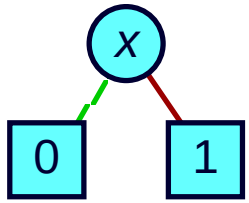
1. It does not contain a node v with high (v) = low (v)
2. There does not exists a pair of nodes u, v such that the sub-OBDDs rooted in u and v are isomorphic

Example Functions

Constants

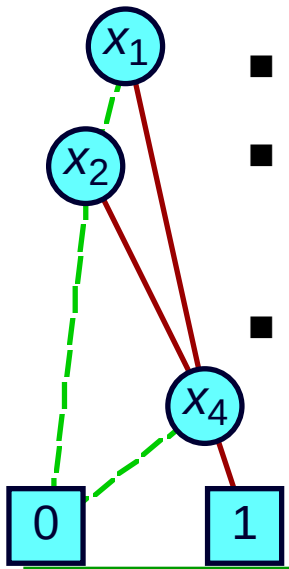
- 0 — **Unique unsatisfiable function**
- 1 — **Unique tautology**

Variable



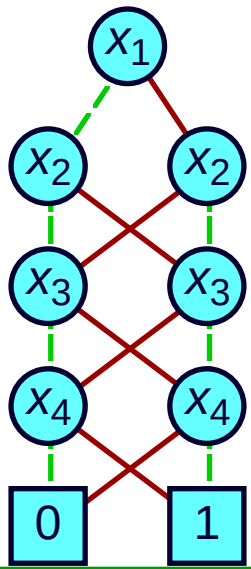
Treat variable as function

Typical Function



- $(x_1 \vee x_2) \wedge x_4$
- **No vertex labeled x_3**
  - independent of x_3
- **Many subgraphs shared**

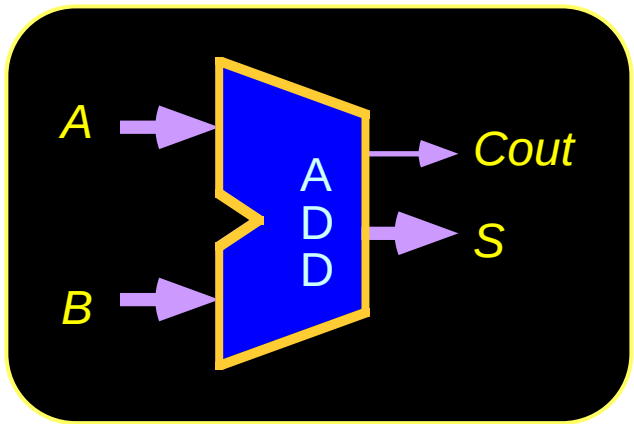
Odd Parity

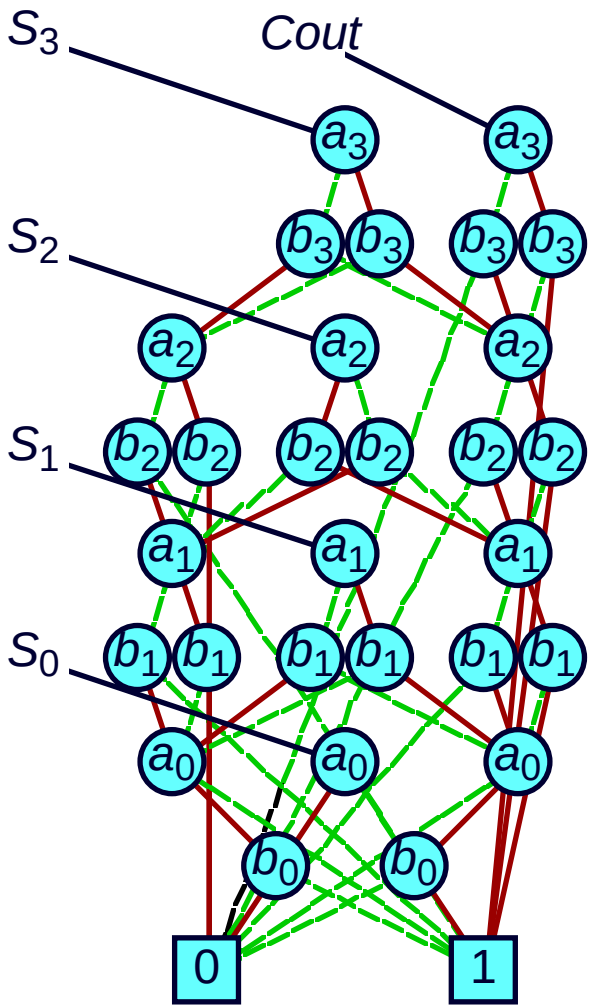


Linear representation

Representing Circuit Functions

- Functions
  - All outputs of 4-bit adder
  - Functions of data inputs



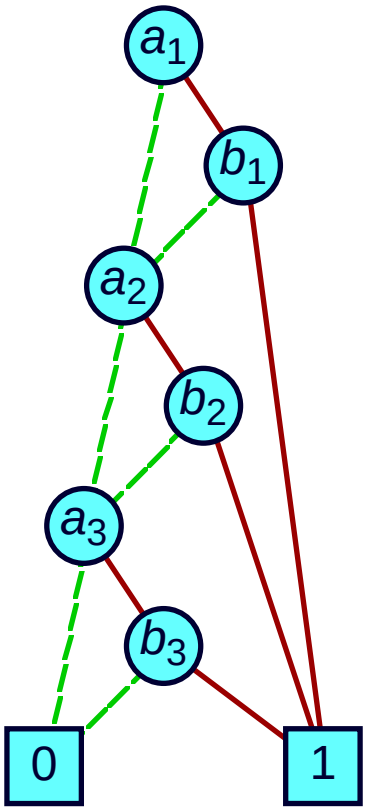


- Shared Representation
  - Graph with multiple roots
  - 31 nodes for 4-bit adder
  - 571 nodes for 64-bit adder
  - ✉ *Linear growth*

Effect of Variable Ordering

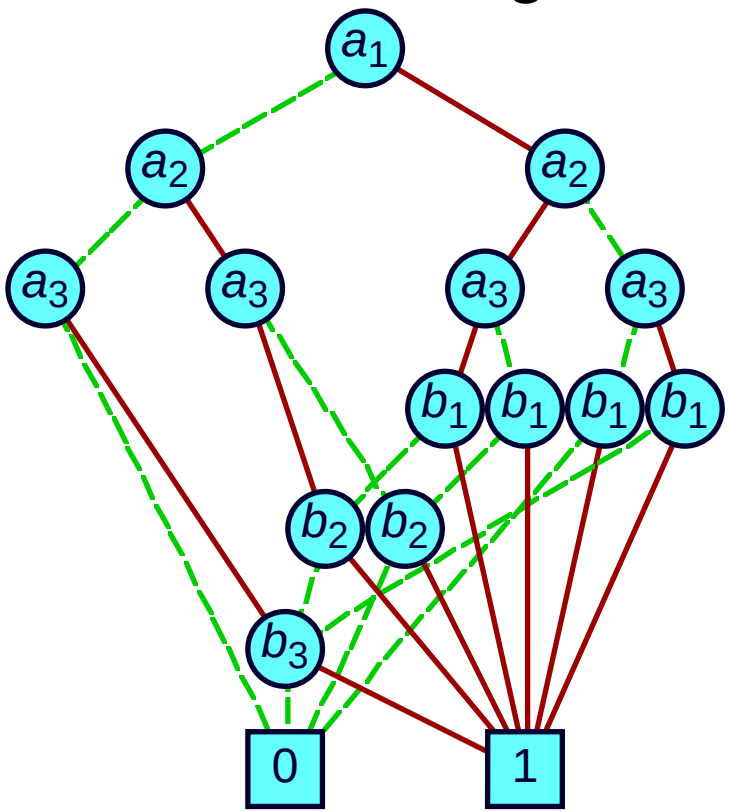
$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$$

Good Ordering



Linear Growth

Bad Ordering



Exponential Growth

Selecting Good Variable Ordering

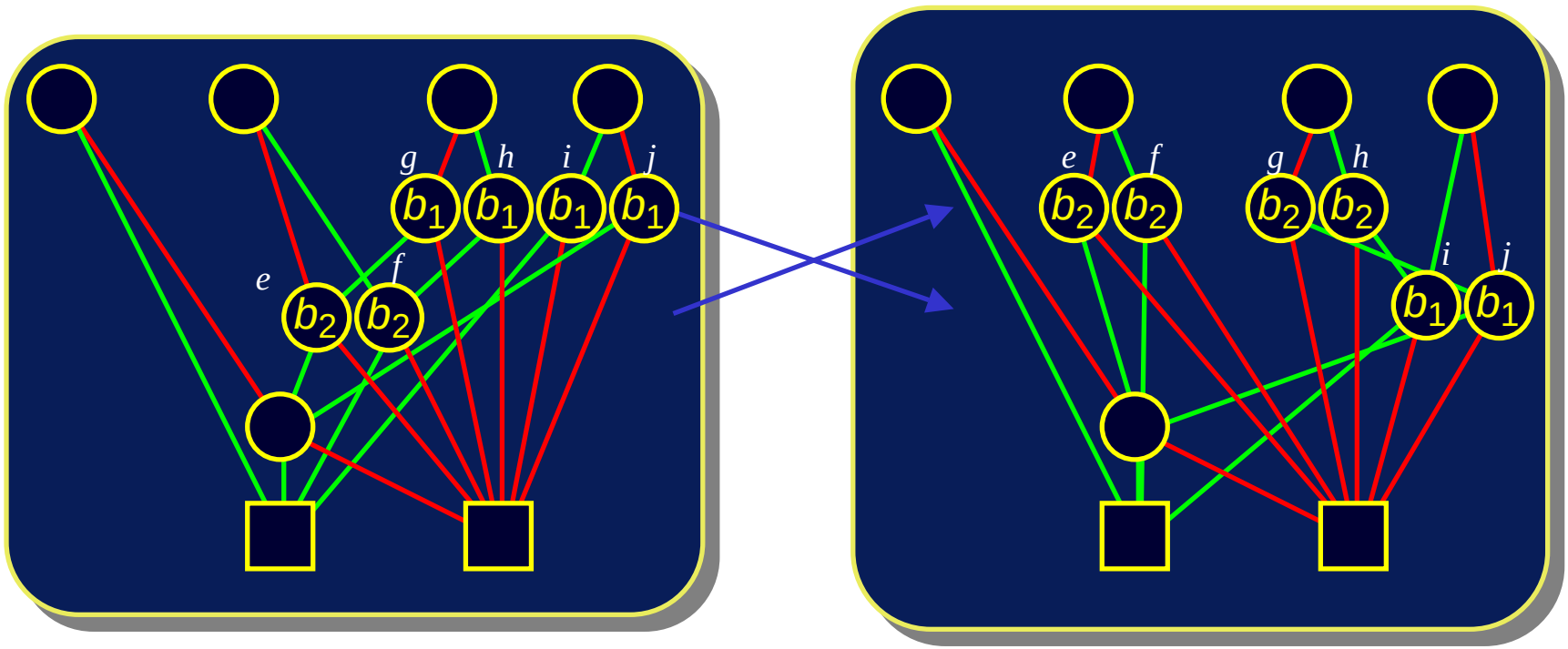
- Intractable Problem
  - Even when problem represented as OBDD
    - i.e., to find optimum improvement to current ordering
- Application-Based Heuristics
  - Exploit characteristics of application
  - e.g., Ordering for functions of combinational circuit
    - Traverse circuit graph depth-first from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

Swapping Adjacent Variables

- Localized Effect
  - Add / delete / alter only nodes labeled by swapping variables
  - Do not change any incoming pointers

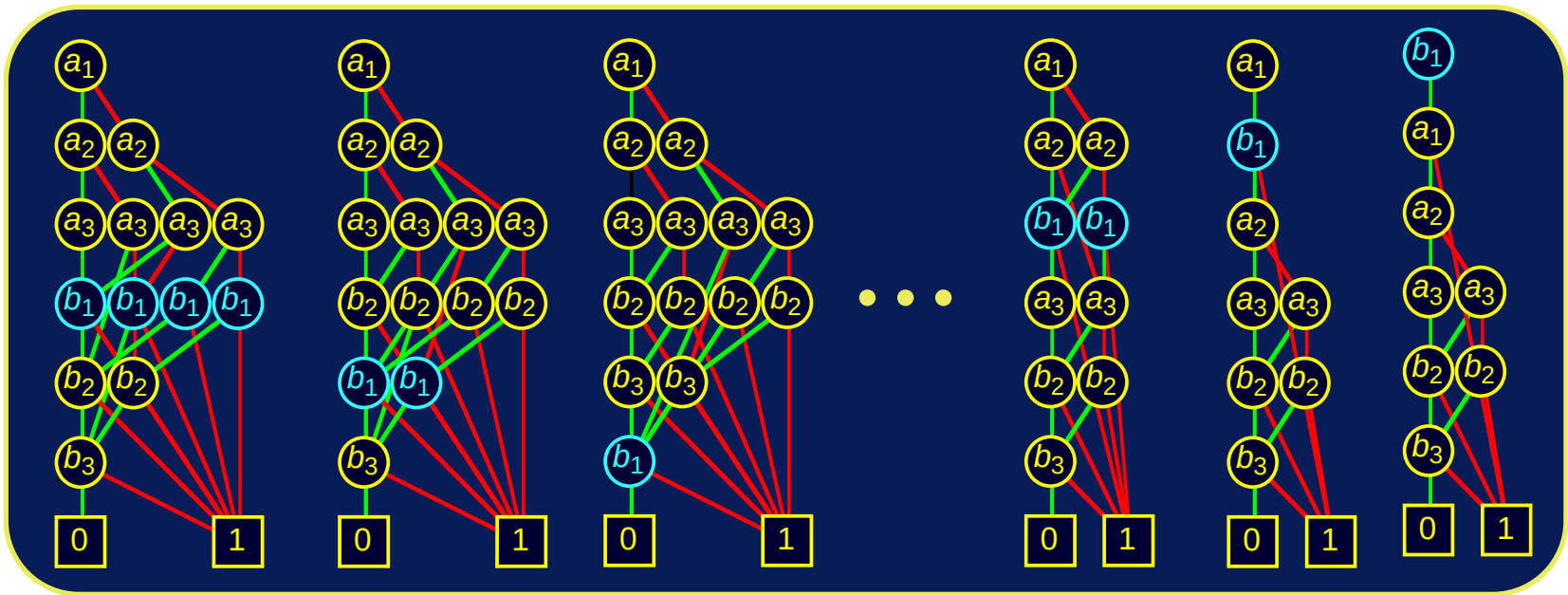


Dynamic Variable Reordering

- Richard Rudell, Synopsys
- Periodically Attempt to Improve Ordering for All BDDs
  - Part of garbage collection
  - Move each variable through ordering to find its best location
- Has Proved Very Successful
  - Time consuming but effective
  - Especially for sequential circuit analysis

Dynamic Reordering By Sifting

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



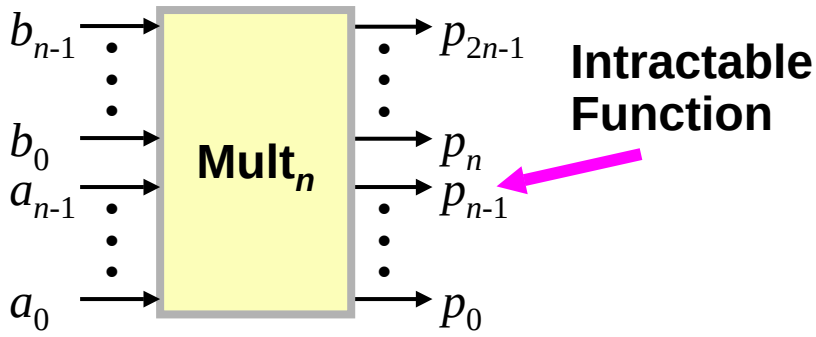
Sample Function Classes

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

Lower Bound for Multiplication

- Bryant, 1991
- Integer Multiplier Circuit
  - n-bit input words A and B
  - $2n$-bit output word P
- Boolean function
  - Middle bit (n-1) of product
- Complexity
  - Exponential OBDD for all possible variable orderings



Actual Numbers

- **40,563,945 BDD nodes to represent all outputs of 16-bit multiplier**
- **Grows 2.86x per bit of word size**

Thank you