



# Introduction to Binary Decision Diagram

Prof. Chien-Nan Liu

TEL: 03-5712121 ext:31211

Email: jimmyliu@nctu.edu.tw

1

## Outlines

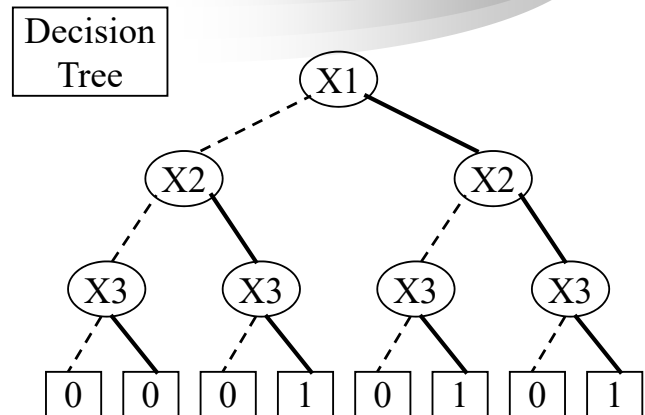


- Representing Boolean Functions
  - Decision graph structure
  - Reduction to canonical form
  - Effect of variable ordering
  - Variants to reduce storage
- Algorithms
  - General framework
  - Basic operations
    - » Restriction (Cofactor)
    - » If-Then-Else
  - Derived operations
  - Computing functional properties

2

# Decision Structures

Truth Table	X1	X2	X3	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

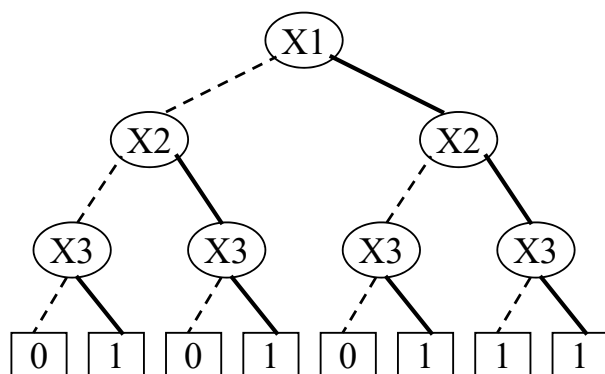


- Vertex represents decision
- Follow dashed line for value 0
- Follow solid line for value 1
- Function value determined by leaf value

3

## Binary Decision Diagram (BDD)

$$f = x_1x_2 + x_3$$



----- : 0

———— : 1

terminal node :

- attribute
  - value(v) = 0
  - value(v) = 1

nonterminal node :

- index(v) = i
- two children
  - low(v)
  - high(v)

4

# BDD

A BDD graph which has a vertex  $v$  as root corresponds to the function  $F_v$  :

(1) If  $v$  is a terminal node :

a) if  $\text{value}(v)$  is 1, then  $F_v = 1$

b) if  $\text{value}(v)$  is 0, then  $F_v = 0$

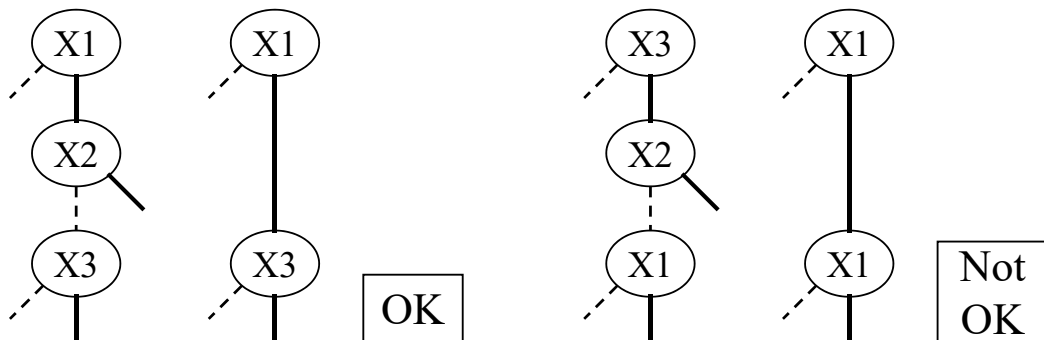
(2) If  $F$  is a nonterminal node (with  $\text{index}(v) = i$ )

$$F_v(x_1, \dots, x_n) = x_i' F_{\text{low}(v)}(x_{i+1}, \dots, x_n) + x_i F_{\text{high}(v)}(x_{i+1}, \dots, x_n)$$

5

## Variable Ordering

- Assign arbitrary total ordering to variable  
e.g.  $X1 < X2 < X3$
- Variable must appear in ascending order along all paths

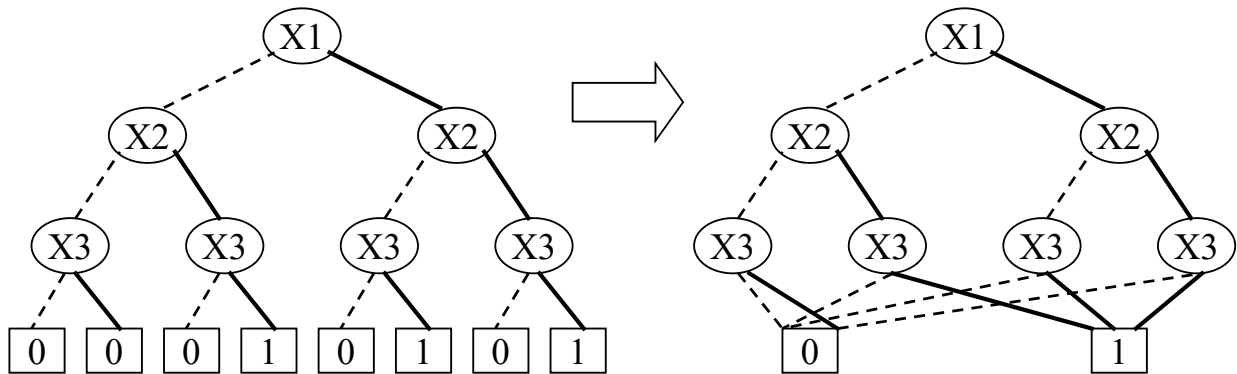
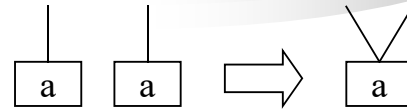


- Properties
  - No conflicting variable assignments along path
  - Simplifies manipulation

6

# Reduction Rule #1

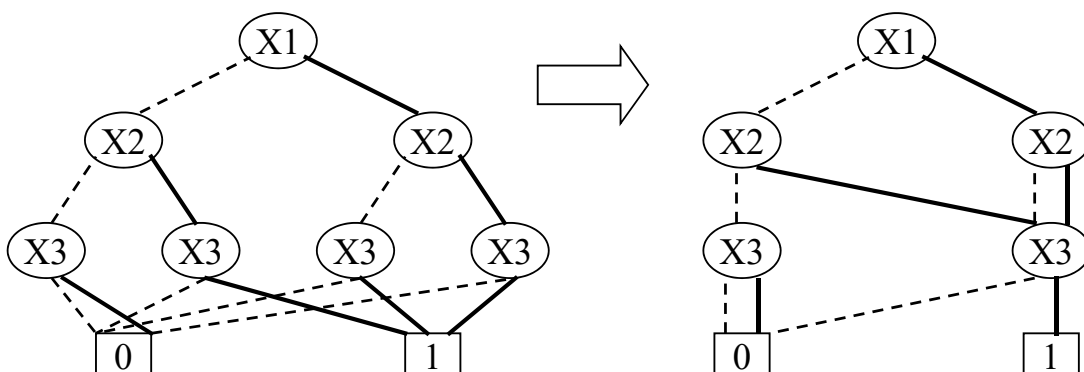
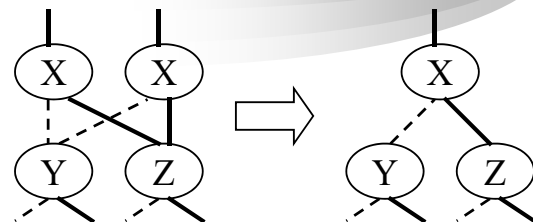
- Merge equivalent leaves



7

# Reduction Rule #2

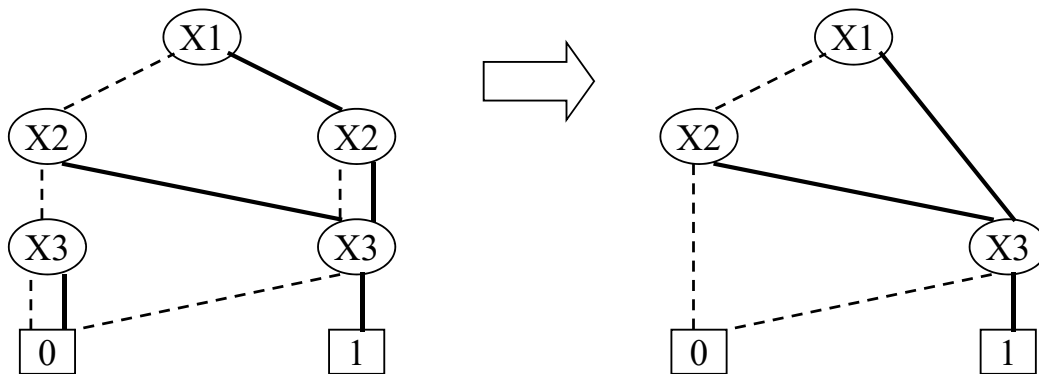
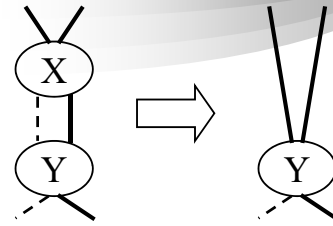
- Merge isomorphic nodes



8

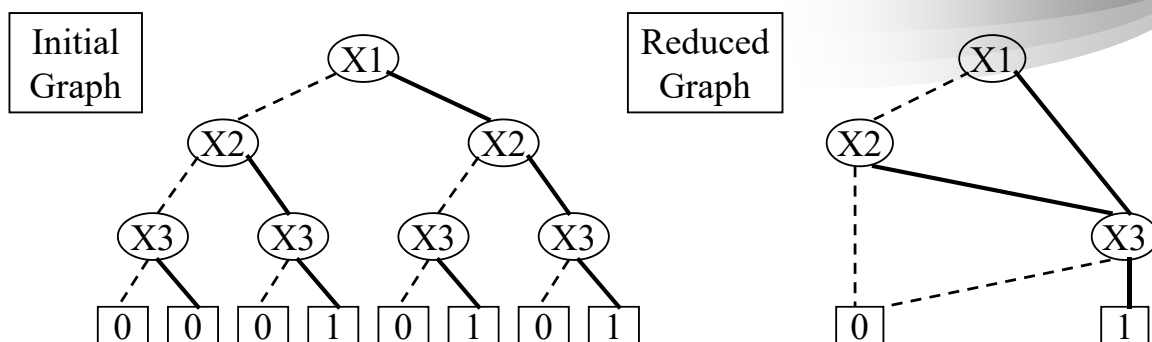
## Reduction Rule #3

- Eliminate Redundant Tests



9

## Example ROBDD



- Canonical representation of Boolean function for given variable ordering
  - Two functions equivalent iff graphs isomorphic
    - » can be tested in linear time
  - Desirable property : The simplest form is canonical

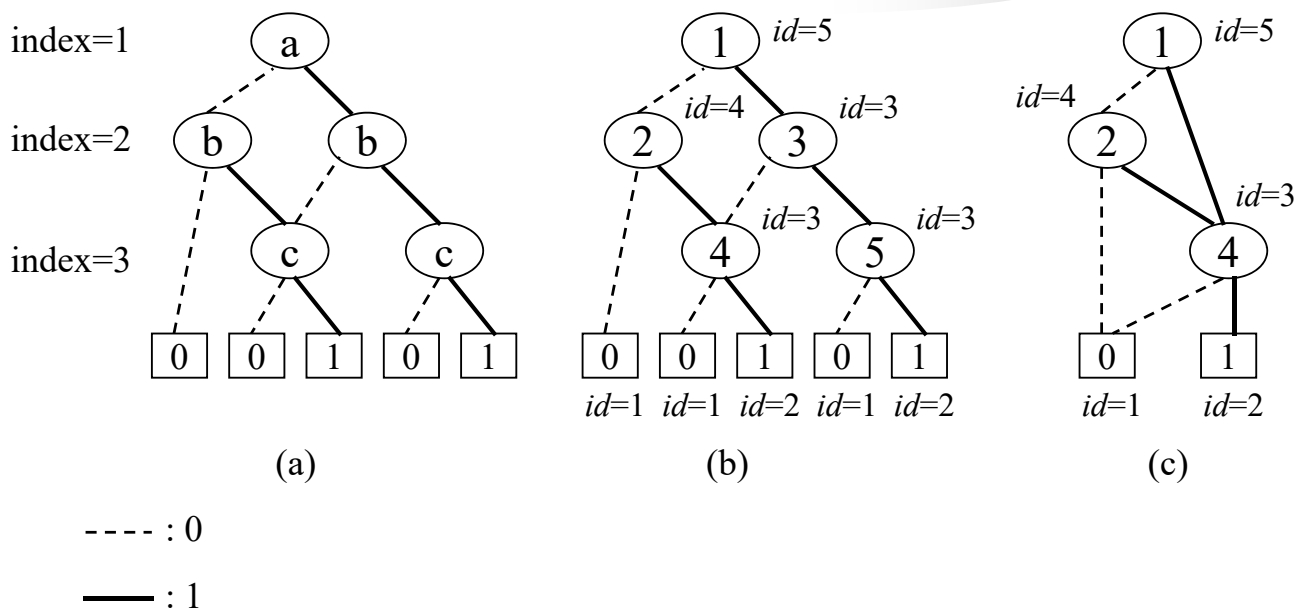
10

# Reduce

- Visit OBDD bottom up and label each vertex with an identifier
- Redundancy
  - if  $\text{id}(\text{low}(v)) = \text{id}(\text{high}(v))$ , then vertex  $v$  is redundant  
 $\Rightarrow \text{set id}(v) = \text{id}(\text{low}(v))$
  - if  $\text{id}(\text{low}(v)) = \text{id}(\text{low}(u))$  and  $\text{id}(\text{high}(v)) = \text{id}(\text{high}(u))$ , then set  $\text{id}(v) = \text{id}(u)$
- A different identifier is given to each vertex at level  $i$
- Terminated when root is reached
- An ROBDD is identified by a subset of vertices with different identifiers

11

# Reduce



12

# Construct ROBDD Directly

- Using a hash table called unique table
  - Contain a key for each vertex of an OBDD
  - Key : (variable, right children, left children)
  - Constructed bottom up
  - Each key uniquely identify the specific function
  - Look up the table can determine if another vertex in the table implements the same function

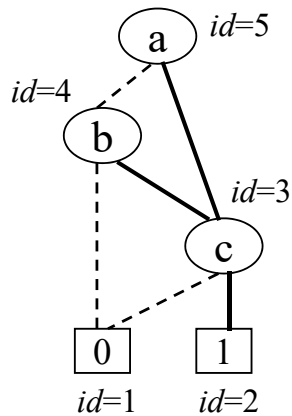
13

## The Unique Table

- Represent an ROBDD
- A strong canonical form
- Check equivalence of two Boolean functions by comparing the corresponding identifiers
- Can represent multiple-output functions

14

# Multi-Rooted ROBDD



Unique table

**Key**

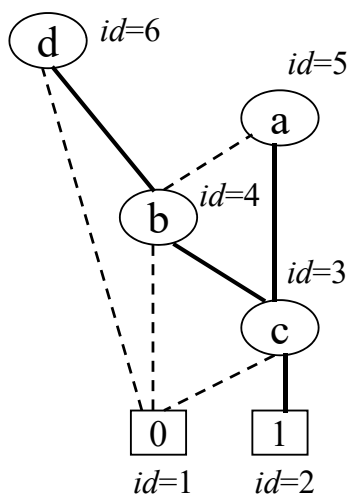
Identifier	Variable	Right child	Left child
5	a	3	4
4	b	3	1
3	c	2	1

$$f = (a+b) c$$

variable order (a, b, c)

15

# Multi-Rooted ROBDD



$f$  is constructed first and is associated with  $id=5$   
 $g : id=6$

Unique table

**Key**

Identifier	Variable	Right child	Left child
6	d	4	1
5	a	3	4
4	b	3	1
3	c	2	1

$$f = (a+b) c$$

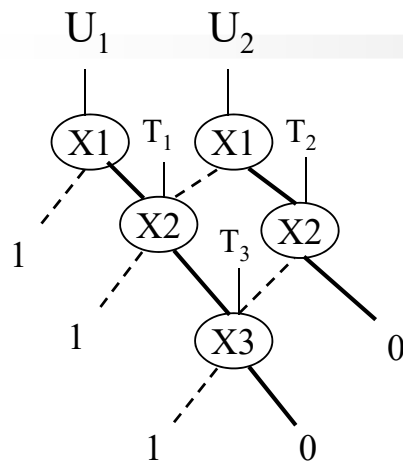
$$g = b c d$$

variable order (d, a, b, c)

16





# The Unique Table



# Hash Table Mapping

$$(X1, T1, 1) \rightarrow U1$$
$$(X1, T2, T1) \dashrightarrow U2$$
$$(X2, T3, 1) \rightarrow T1$$
$$(X2, 0, T3) \rightarrow T2$$

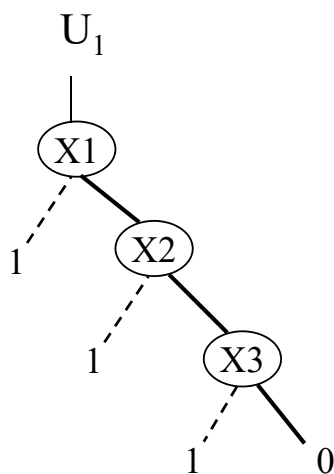
(X3, 0 , 1 ) --> T3

true 
 false

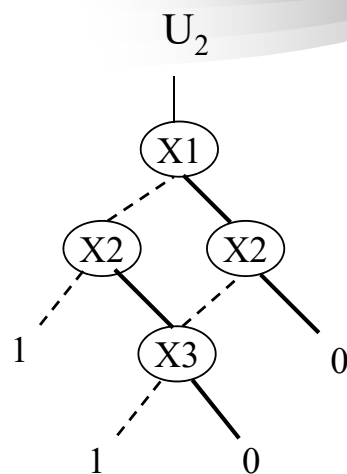
- **Unique table** : hash table mapping  $(X_i, G, H)$  into a node in the DAG
  - before adding a node to the DAG, check to see if it already exists
  - avoids creating two nodes with the same function
  - strong canonical form : pointer equality determines function equality

17

## Non-Shared ROBDD

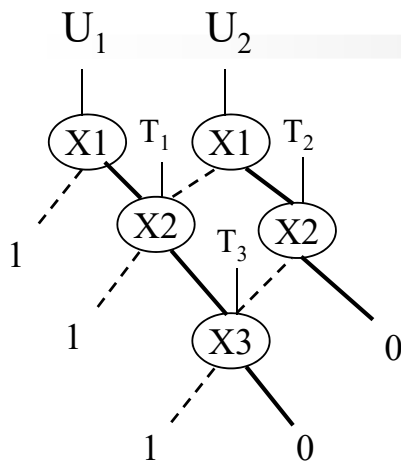


$$U1 = X1' + X2' + X3'$$



$$U_2 = X_1'X_2' + X_1'X_3'$$

# Multi-Rooted (Shared) ROBDD



$$U_1 = X_1' + X_2' + X_3' = (X_1, T_1, 1)$$

$$U_2 = X_1'X_2' + X_1'X_3' = (X_1, T_2, T_1)$$

$$T_1 = X_2' + X_3' = (X_2, T_3, 1)$$

$$T_2 = X_2'X_3' = (X_2, 0, T_3)$$

$$T_3 = X_3' = (X_3, 0, 1)$$

$$0 = (X_\infty, 0, 0)$$

$$1 = (X_\infty, 1, 1)$$

External functions

User functions

Internal functions

- A DAG node F is represented by a tuple  $(X_i, G, H)$ 
  - $X_i$  is called the top variable of F
  - node  $(X_i, G, H)$  represents the function  $ite(X_i, G, H) = X_iG + X_i'H$
- DAG contains both external and internal functions

19

## Separated vs. Shared

- Separated
  - 51 nodes for 4-bit adder
  - 12481 nodes for 64-bit adder
  - Quadratic growth
- Shared
  - 31 nodes for 4-bit adder
  - 571 nodes for 64-bit adder
  - Linear growth

20

# Maintaining Shared ROBDD

- Storage Model
  - Single, multiple-rooted DAG
  - Function represented by pointer to node in DAG
  - Maintain Unique (hash) table to keep canonical
- Storage Management
  - User cannot know when storage for node can be freed
  - Must implement automatic garbage collection
- Algorithmic Efficiency
  - Functions equivalent iff pointer equal
    - » if (p1 == p2) ...
  - Can test in constant time

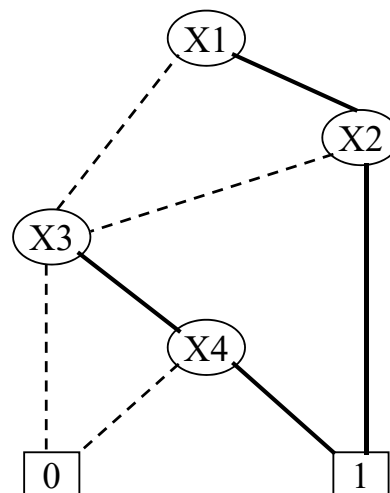
21

## Ordering Effects

- The size of ROBDD depends on the ordering of variables

ex :  $x_1x_2 + x_3x_4$

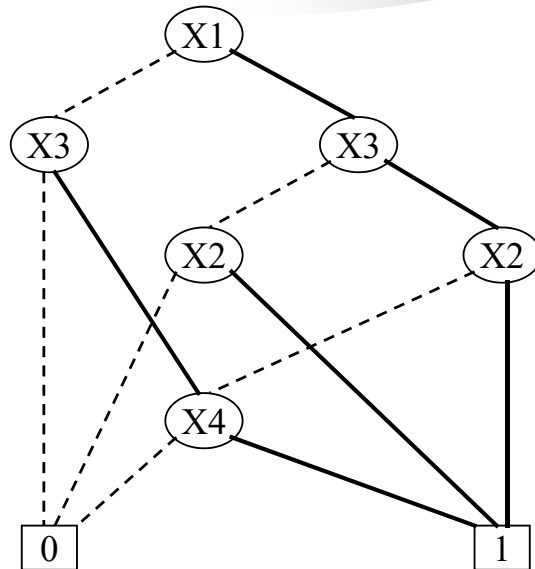
$x_1 < x_2 < x_3 < x_4$



22

## Ordering Effects (cont'd)

$$x_1 < x_3 < x_2 < x_4$$



23

## 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 ROBDD representations
  - Algorithms remain practical for up to 100,000 vertex ROBDD
  - Heuristic ordering methods generally satisfactory

24

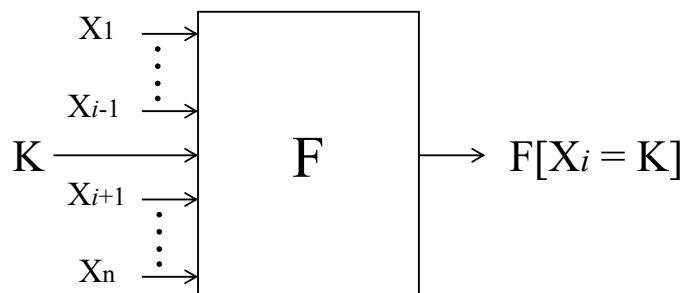
# Symbolic Manipulation

- Strategy
  - Represent data as set of ROBDDs
    - » with identical variable orderings
  - Express solution method as sequence of symbolic operations
  - Implement each operation by ROBDD manipulation
- Algorithmic Properties
  - Arguments are ROBDDs with identical variable orderings
  - Result is ROBDD with same ordering
  - “Closure Property”
- Two Basic Operations
  - Restriction
  - If-Then-Else

25

## Restriction Operation

- Concept
  - Effect of setting function argument  $X_i$  to constant  $K(0,1)$
  - Also called Cofactor operation



- Implementation
  - Depth-first traversal
  - Complexity near-linear in argument graph size

26

# Restriction Algorithm

Restrict (F, x, k)

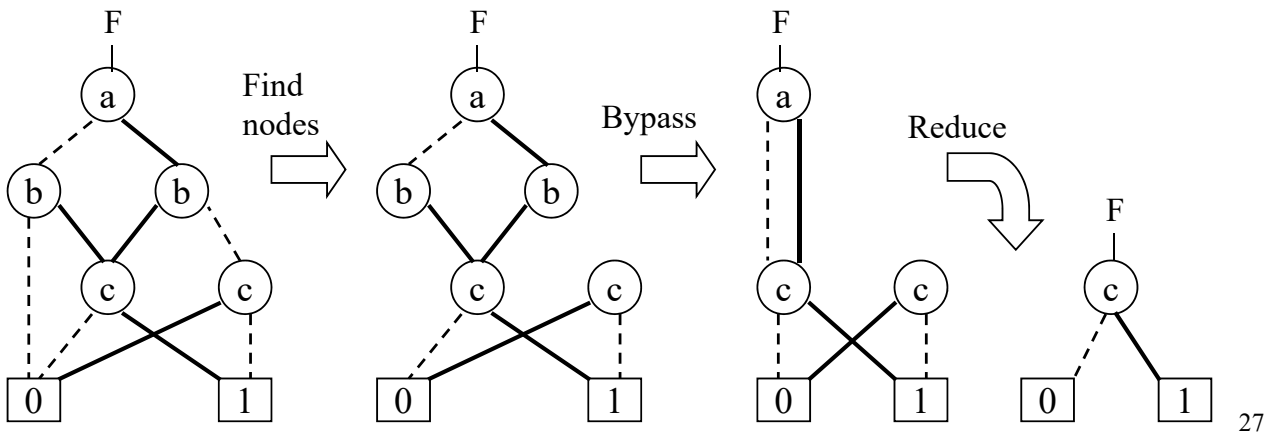
Bypass any nodes for variable x

Choose Hi child for k = 1

Choose Lo child for k = 0

Reduce result

e.g. Restrict variable b to 1

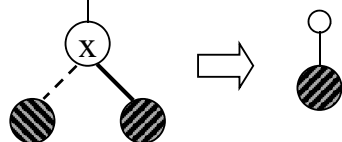


27

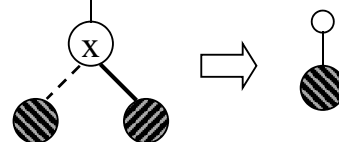
## Special cases of Restriction

- Case 1 : Restrict on root node variable

Restrict (  $\circ$ , x, 1 )



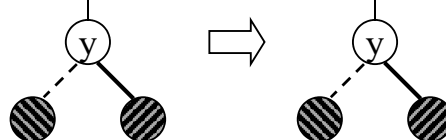
Restrict (  $\circ$ , x, 0 )



- Case 2 : Restrict on variable less than root node

– e.g.  $x < y$

Restrict (  $\circ$ , x, 1 )



28

# If-Then-Else Operation

- Concept
  - Basic technique for building ROBDD from network or formula
- Argument **I** (if), **T** (then), **E** (else)
  - Functions over variables **X**
  - Represented as ROBDDs
- Result
  - ROBDD representing composite function
  - $IT + I'E$
- Implementation
  - combination of depth-first traversal and dynamic programming
  - Worst case complexity : product of argument graph sizes

29

# If-Then-Else Algorithm

- Recursive Formulation
$$\text{ITE}(I, T, E) = x \text{ITE}(I[x=1], T[x=1], E[x=1]) + x' \text{ITE}(I[x=0], T[x=0], E[x=0])$$
- General Algorithm
  - Select top root variable **x** of **I**, **T** and **E**
  - Compute restrictions
    - » Guaranteed to be one of special cases
  - Apply recursively to get results **Lo** and **Hi**
  - Still remain canonical form
- Termination Conditions
  - $I = 1 \implies \text{Return } T$
  - $I = 0 \implies \text{Return } E$
  - $T = 1, E = 0 \implies \text{Return } I$
  - $T = E \implies \text{Return } T$

30

# An ITE Example

- Given  $f = ab + bc + ac$ ,  $g = c$  under the order  $a < b < c$

ITE ( $f$ ,  $g$ , 0)

= ITE[  $a$ , ITE(  $f(a=1)$ ,  $g(a=1)$ , 0), ITE(  $f(a=0)$ ,  $g(a=0)$ , 0)]

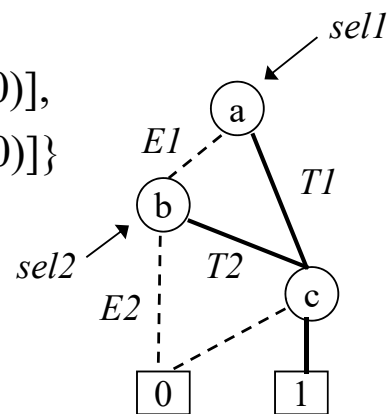
= ITE[  $a$ , ITE(  $b+bc+c$ ,  $c$ , 0), ITE(  $bc$ ,  $c$ , 0)]

= ITE{  $a$ , ITE[  $b$ , ITE(1,  $c$ , 0), ITE( $c$ ,  $c$ , 0)],  
ITE[  $b$ , ITE( $c$ ,  $c$ , 0), ITE(0,  $c$ , 0)]}

= ITE[  $a$ , ITE( $b$ ,  $c$ ,  $c$ ), ITE( $b$ ,  $c$ , 0)]

= ITE[  $a$ ,  $c$ , ITE( $b$ ,  $c$ , 0)]

$sel1 \leftarrow$   $\downarrow$   $T1$   $E1=sel2$   $\downarrow$   $T2$



31

## Algorithmic Issues & Derived Operations

- Efficiency
  - Maintain computed table and unique table to increase efficiency
  - Worst case complexity product of graph sizes for I, T, E
- Derived operations
  - Express as combination of If-Then-Else and Restrict
  - Preserve closure property
    - » Result is a ROBDD with the same variable ordering

32



# Detailed ITE Algorithm

```

ITE(f, g, h) {
  if (terminal case)
    return (r = trivial result) ;
  else {
    /* exploit previous information */
    if (computed table has entry {(f, g, h), r} )
      return (r from computed table) ;
    else {
      x = top variable of f, g, h ;
      t = ITE(fx, gx, hx) ;
      e = ITE(fx', gx', hx') ;
      if (t == e)
        /* children with isomorphic OBDDs */
        return (t) ;
      r = find_or_add_unique_table(x, t, e) ; /* add r to unique table if not present */
      Update computed table with {(f, g, h), r} ;
      return (r) ;
    }
  }
}

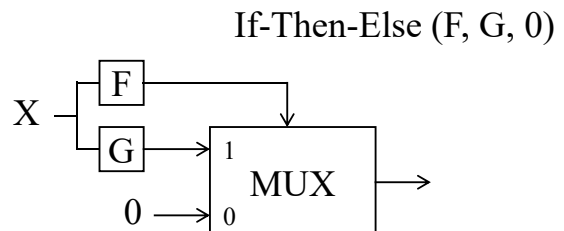
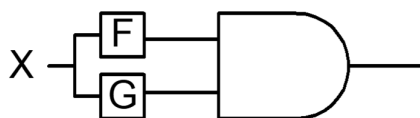
```

33

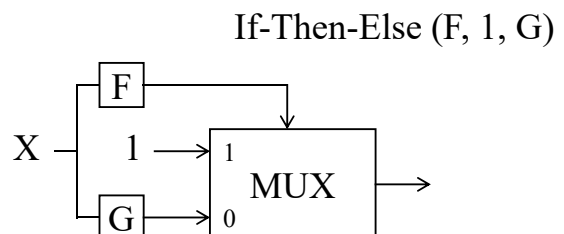
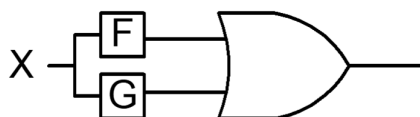
## Derived Algebraic Operations

- Other common operations can be expressed in terms of If-Then-Else

AND (F, G)



OR (F, G)



34

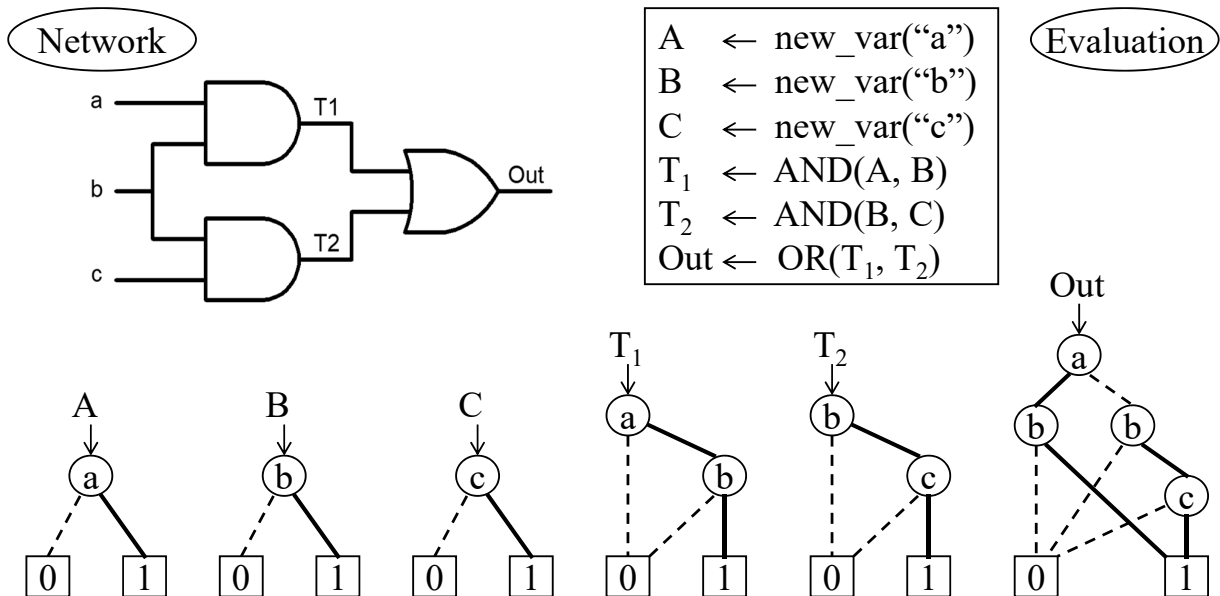
# ITE Operators

Operator	Equivalent <i>ite</i> form
0	0
$f \cdot g$	$ite(f, g, 0)$
$f \cdot g'$	$ite(f, g', 0)$
$f$	$f$
$f' \cdot g$	$ite(f, 0, g)$
$g$	$g$
$f \oplus g$	$ite(f, g', g)$
$f + g$	$ite(f, 1, g)$
$(f + g)'$	$ite(f, 0, g')$
$(f \oplus g)'$	$ite(f, g, g')$
$g'$	$ite(g, 0, 1)$
$f + g'$	$ite(f, 1, g')$
$f'$	$ite(f, 0, 1)$
$f' + g$	$ite(f, g, 1)$
$(f \cdot g)'$	$ite(f, g', 1)$
1	1

35

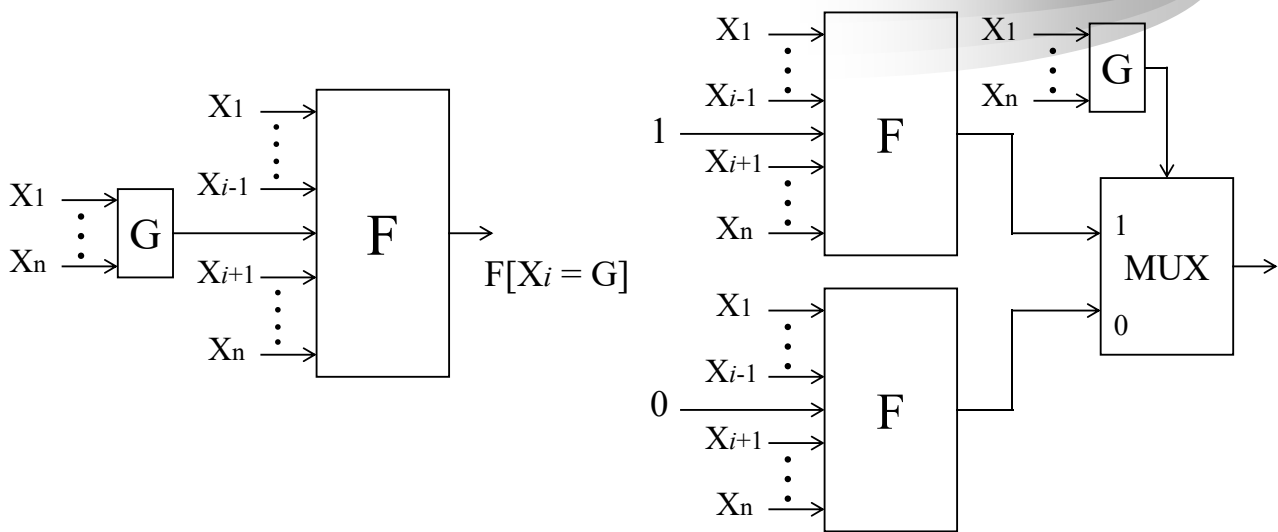
## Generating ROBDD from Network

- Task : Represent output functions of gate network as ROBDDs



36

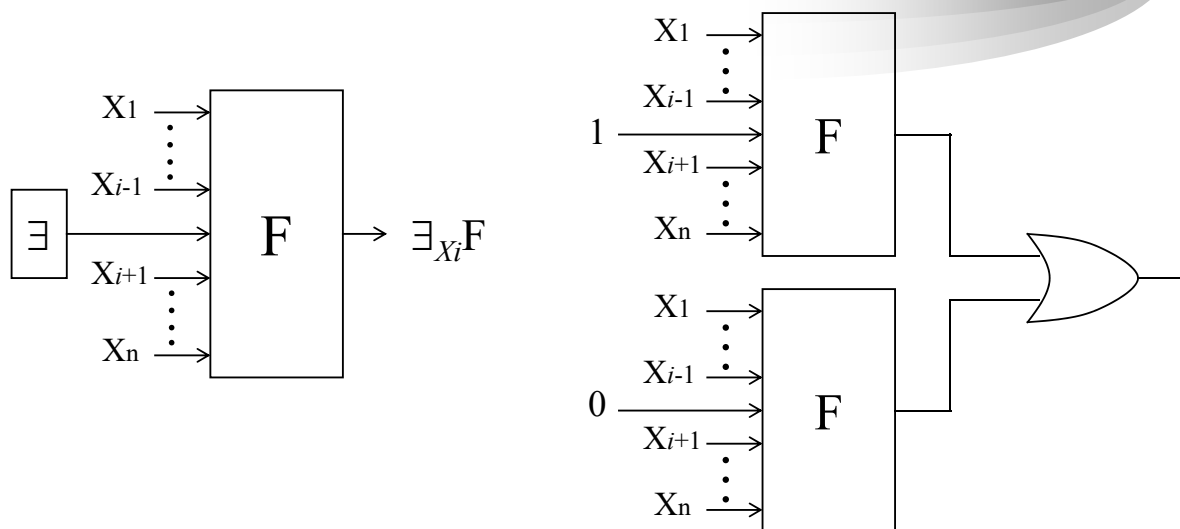
# Functional Composition



- Create new function by composing functions  $F$  and  $G$
- Useful for composing hierarchical modules

37

# Variable Qualification



- Eliminate dependency on some argument through qualification

38

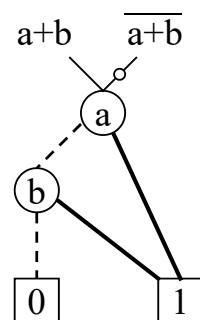
# Variants & Optimizations

- Concept
  - Refinements to ROBDD representation
  - Do not change fundamental properties
- Objective
  - Reduce memory requirement
  - Improve algorithmic efficiency
  - Make commonly performed operations faster
- Common Optimizations
  - Share nodes among multiple functions
  - Negated arcs

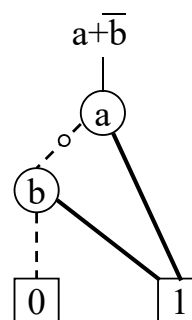
39

## Negation Arcs

- Concept
  - Dot on arc represents complement operator
    - » Invert function value
  - Can appear internal or external arc



external

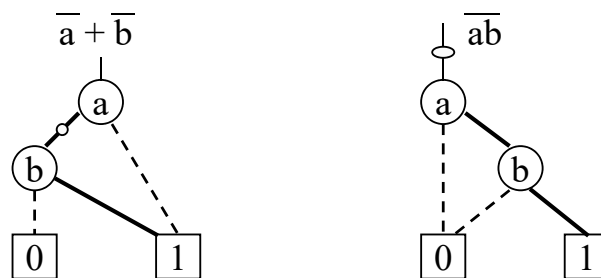


internal

40

# Effect of Negation Arcs

- Storage Savings
  - At most 2X reduction in numbers of nodes
- Algorithmic Improvement
  - Can complement function in constant time
- Problem
  - Negation arc allow multiple representations of a function



- Modify algorithms with restricted conversions for use of negative arcs

41

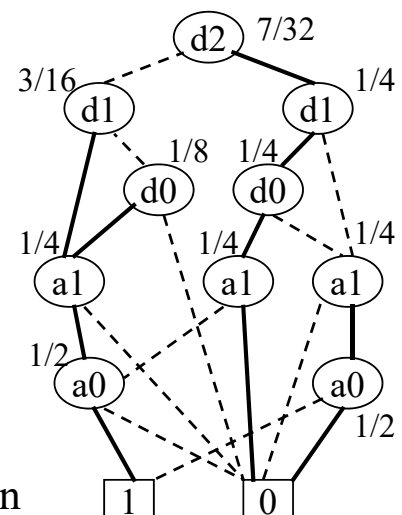
# Density Computation

- Definition
  - $p(F)$  : fraction of variable assignments for which  $F = 1$

- Applications
  - Testability measures
  - Probability computations

- Recursive Formulation
  - $p(F) = [ p( F[x=1] ) + p( F[x=0] ) ] / 2$

- Computation
  - Compute bottom-up, starting at leaves
  - At each node, average density of children



42

# Characteristic Function

Let  $E$  be a set and  $A \subseteq E$

The characteristic function of  $A$  is the function

$$X_A : E \rightarrow \{0, 1\}$$

$$X_A(x) = 1 \text{ if } x \in A$$

$$X_A(x) = 0 \text{ if } x \notin A$$

Ex :

$$E = \{1, 2, 3, 4\}$$

$$A = \{1, 2\}$$

$$X_A(1) = 1$$

$$X_A(3) = 0$$

43

# Characteristic Function

Given a Boolean function

$$f : B^n \rightarrow B^m$$

the mapping relation denoted as  $F \subseteq B^n \times B^m$  is defined as

$$F(x, y) = \{ (x, y) \in B^n \times B^m \mid y = f(x) \}$$

The characteristic function of a function  $f$  is defined for  $(x, y)$  s.t.  $X_f(x, y) = 1$  iff  $(x, y) \in F$

44

# Characteristic Function

Ex :  $y = f(x_1, x_2) = x_1 + x_2$

$F_y(x_1, x_2, y) =$

$x_1$	$x_2$	$y$		$x_1$	$x_2$	$y$	$F$
0	0	0	→	0	0	0	1
0	1	1	→	0	0	1	0
1	0	1	→	0	1	0	0
1	1	1	→	0	1	1	1
			→	1	0	0	0
			→	1	0	1	1
			→	1	1	0	0
			→	1	1	1	1

45

## Summary

- ROBDD
  - Reduced graph representation of Boolean Function
  - Canonical for given variable ordering
  - Size sensitive to variable ordering
- Algorithmic Principles
  - Operations maintain closure property
    - » Result ROBDD with same ordering as arguments
    - » Can perform further operations on results
  - Limited set of basic operations to implement
    - » Restrict, If-Then-Else
    - » Other operations defined in terms of basic operations

46