EE6094
CAD for VLSI Design

Chapter 4
Boolean Representation

Spring 2024
Andy, Yu-Guang Chen
Assistant Professor, Department of EE
National Central University
andyygchen@ee.ncu.edu.tw

Outline

- Binary system representations
- Definitions of BDDs, OBDDs and ROBDDs
- Logic operations on BDDs
- The ITE operator
- Variable ordering (static and dynamic)

Basic Definitions

- Let B = {0,1} Y = {0,1,2}
  - A logic function f in *n* inputs $x_1, x_2, \ldots x_n$ and *m* outputs $y_1, y_2, \ldots y_m$ is a function

$$f : B^n \longrightarrow Y^m$$

$$X = [x_1, x_2, \ldots, x_n] \in B^n \quad \text{is the input}$$

$$Y = [y_1, y_2, \ldots, y_m] \in Y^m \quad \text{is the output}$$

  - m=1 → a single output function
  - m>1 → a multiple output function

Basic Definitions

- For each component f_i, i = 1,2, ...,m, define
  - ON_SET: set of input values x such that $f_i(x) = 1$
  - OFF_SET: set of input values x such that $f_i(x) = 0$
  - DC_SET: set of input values x such that $f_i(x) = 2$
- Completely specified function: DC_SET = ϕ, $\forall f_i$
- Incompletely specified function: DC_SET $\neq \phi$, for some f_i

Boolean Representations

- Truth table representation

Full adder

X	Y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

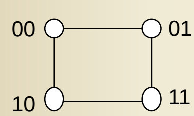
- A multiple output function
- Sum
  - on-set = {(0 0 1), (0 1 0), (1 0 0), (1 1 1)}
  - off-set = {(0 0 0), (0 1 1), (1 0 1), (1 1 0)}
- A completely specified function

Boolean Representations

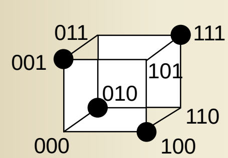
- Geometrical representation

1 variable

2 variables



3 variables



sum : on-set ={(0 0 1),(0 1 0),(1 0 0),(1 1 1)}

off-set ={(0 1 1),(1 0 1),(1 1 0),(0 0 0)}

Boolean Representations

- Algebraic representations
  - Canonical sum of minterms
    - $C_{out} = x'yC_{in} + xy'C_{in} + xyC_{in}' + xyC_{in}$
  - Reduced sum of products
    - $C_{out} = yC_{in} + xC_{in} + xy$
    - $C_{out} = yC_{in} + xC_{in} + xyC_{in}'$
  - Multi-level representation
    - $C_{out} = C_{in} (x + y) + xy$

Minterms and Maxterms

- Minterms and Maxterms
- A minterm (standard product): an AND term consists of all literals in their normal form or in their complement form
  - For example, two binary variables *x* and *y*
    - *xy, xy', x'y, x'y'*
  - It is also called a standard product
  - *n* variables can be combined to form 2^n minterms
- A maxterm (standard sums): an OR term
  - It is also call a standard sum
  - 2^n maxterms

Minterms and Maxterms

- Each *maxterm* is the complement of its corresponding *minterm*, and vice versa

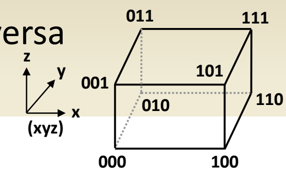


Table 2.3
Minterms and Maxterms for Three Binary Variables

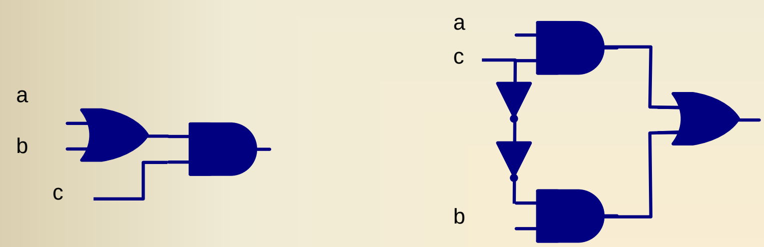
			Minterms		Maxterms	
x	***y***	***z***	**Term**	**Designation**	**Term**	**Designation**
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Standard Forms

- Canonical forms are very seldom with the least number of literals
- Standard forms: the terms that form the function may obtain one, two, or any number of literals
  - Sum of Products (SOP): $F_1 = y' + xy + x'yz'$
  - Product of Sums (POS): $F_2 = x(y' + z)(x' + y + z')$
  - Nonstandard form: $F_3 = AB + C(D + E)$

Boolean Representations

- Logic gate representations



Boolean Representations

- A Binary Decision Diagram (BDD) is a *directed acyclic graph*
  - Graph: set of vertices connected by edges
  - Directed: edges with direction
  - Acyclic: no path in the graph can lead to a cycle

  - Often abbreviated as DAG

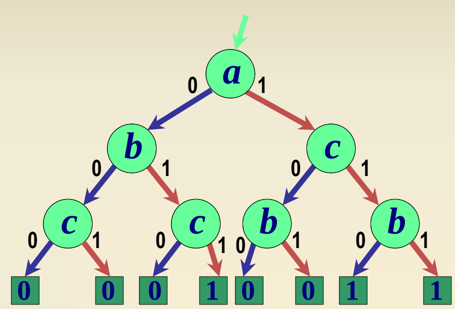
Binary Decision Diagram (BDD)

- A BDD graph which has a vertex v as root corresponds to the function F_v:
  - If v is a terminal node:
    - if value (v) is 1, then $F_v = 1$
    - if value (v) is 0, then $F_v = 0$
  - If F is a non-terminal node (with index(v) = i)
    - $F_v (x_i, \ldots x_n) = x_i' F_{low(v)} (x_{i+1}, \ldots x_n) + x_i F_{high(v)} (x_{i+1}, \ldots x_n)$

BDD Example

- F = (a + b) c

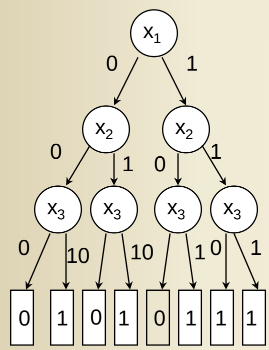
a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



1. Each vertex represents a decision on a variable
2. The value of the function is found at the leaves
3. Each path from root to leaf corresponds to a row in the truth table

BDD Example

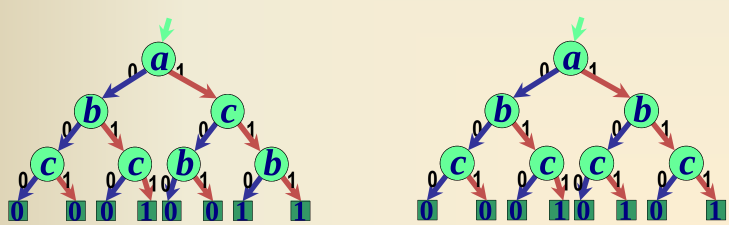
- Binary Decision Diagram (BDD)
- $f = x_1x_2 + x_3$



- Terminal node:
  - Attribute
    - value (v) = 0
    - value (v) = 1
- Non-terminal node:
  - index (v) = i
    - Two children nodes
    - low (v)
    - high (v)
- Evaluate an input vector

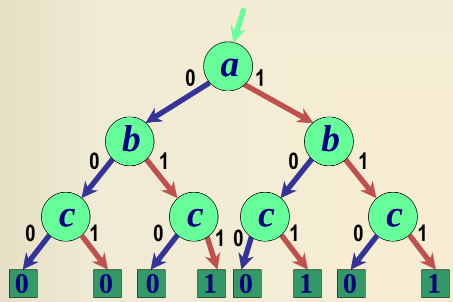
BDD – Observations

- The size of a BDD is as big as a truth table:
  - 1 leaf per row
- Each path from root to leaf evaluates variables in some order
  - but the order is not fixed:



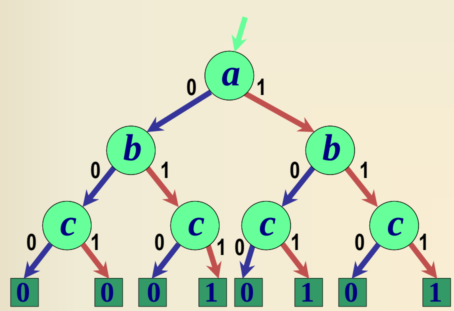
1st idea: Ordered BDD (OBDD)

- Choose arbitrary total ordering on the variables
- Variables must appear in the same order along each path from root to leaves
- Each variable can appear at most once on a path
  - Example: a < b < c



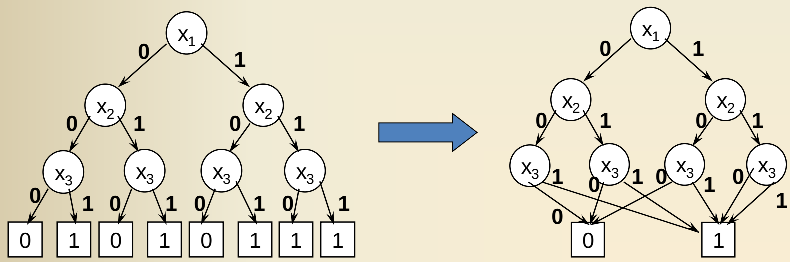
2nd idea: Reduced OBDD (ROBDD)

- Reduced OBDD:
  - No distinct vertices v and v' such that subgraphs rooted by v and v' are isomorphism
  - No vertex v with low (v) = high (v)



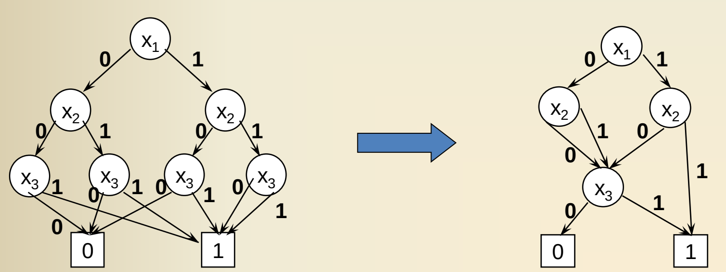
ROBDD Example

- $F = x_1'x_2'x_3+x_1'x_2x_3+x_1x_2'x_3+x_1x_2x_3'+x_1x_2x_3$



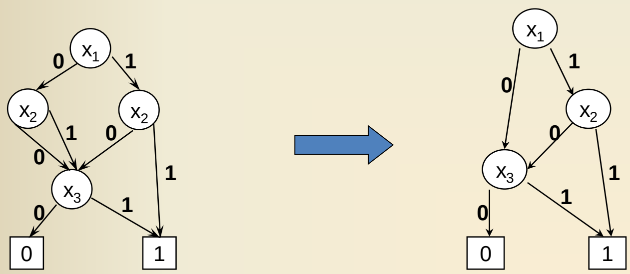
ROBDD Example

- $F = x_1'x_2'x_3+x_1'x_2x_3+x_1x_2'x_3+x_1x_2x_3'+x_1x_2x_3$



ROBDD Example

- $F = x_1x_2+x_3$



BDD Tools

- https://davidkebo.com/cudd
- CUDD stands for Colorado University Decision Diagram
  - A package for the manipulation of
    - Binary Decision Diagrams (BDDs)
    - Algebraic Decision Diagrams (ADDs)
    - Zero-suppressed Binary Decision Diagrams (ZDDs)

BDD Tools

Tutorial 1: Write your first program with CUDD

The basic use of CUDD is easy:

- Initialize a DdManager using Cudd_Init
- Create the DD
- Shut down the DdManager using Cudd_Quit(DdManager* ddmanager)

Sample code for the main program

The program below creates a single BDD variable

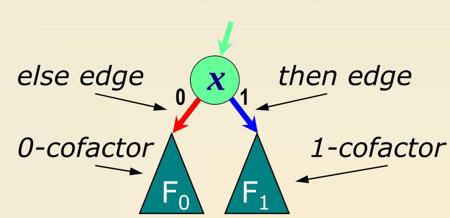
```
int main (int argc, char *argv[])
{
    DdManager *gbm; /* Global BDD manager. */
    char filename[30];
    gbm = Cudd_Init(0,0,CUDD_UNIQUE_SLOTS,CUDD_CACHE_SLOTS,0); /* Initialize a new BDD manager. */
    DdNode *bdd = Cudd_bddNewVar(gbm); /*Create a new BDD variable*/
    Cudd_Ref(bdd); /*Increases the reference count of a node*/
    Cudd_Quit(gbm);
    return 0;
}
```


BDD Semantics

Constant nodes

0 **1**

ITE(x,F1,F0)

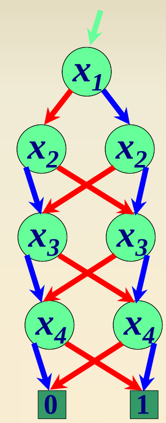


◆ cofactor(F, x): the function you obtain when you substitute 1 for x in F

ROBDD Property

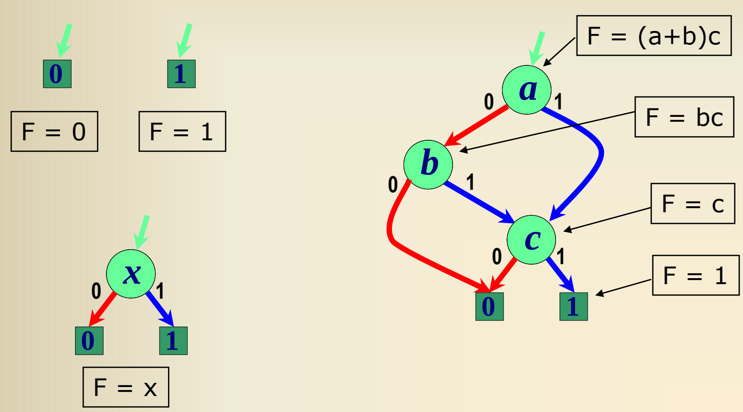
- ROBDDs are canonical
  - For a given variable order
- ROBDDs are more compact than other canonical forms
- ROBDDs size depend on the variable order
  - many useful functions have linear-space representations

Inputs			outputs
W	X	Y	Q = A⊕B⊕C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

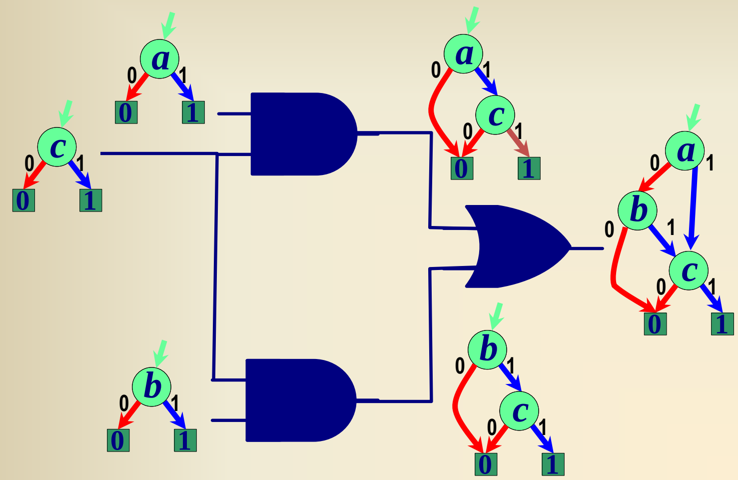


$$F = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$

A Few Simple Functions



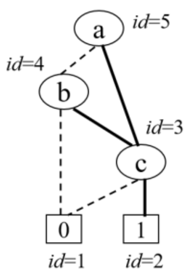
A Network Example



Construct ROBDD Directly

- Using a hash table called unique table
  - Contain a key for each vertex of an OBDD
  - Key: (variable, right child, left child)
  - 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

Construct ROBDD Directly



Unique table

Key

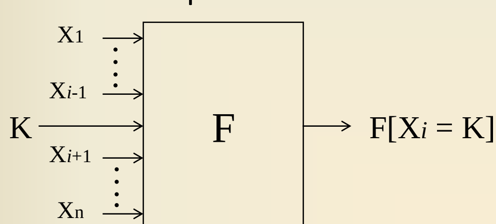
Identifier	Variable	Right child	Left child
5	a	3	4
4	b	3	1
3	c	2	1

f = (a+b) c

variable order (a, b, c)

How to Find Cofactor

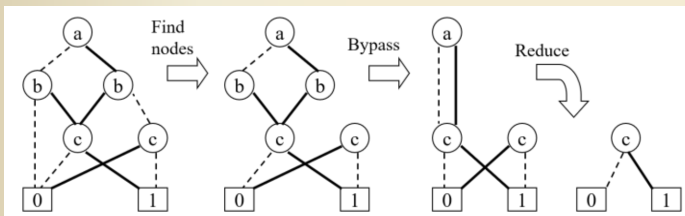
- Restriction Operation
  - Effect of setting function argument Xi to constant K(0,1)
  - Also called Cofactor operation



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

How to Find Cofactor

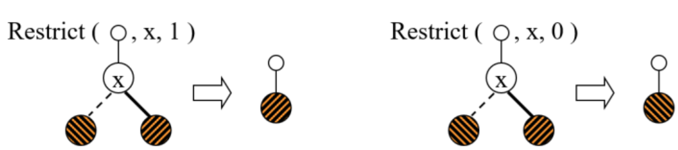
Restrict (F, x, k)
- Bypass any nodes for variable x
  - Choose Hi child for k = 1
  - Choose Lo child for k = 0
- Reduce result



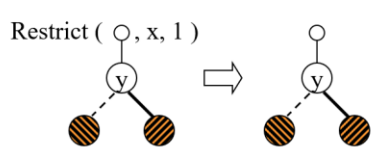
e.g. Restrict variable b to 1

How to Find Cofactor

- Case 1 : Restrict on root node variable



- Case 2 : Restrict on variable whose index is less than root node
  - e.g. x < y

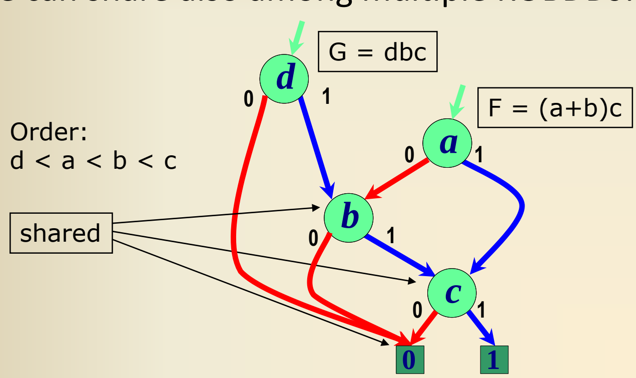


ROBDDs – Why Do We Care?

- Easy to solve some important problems:
  - Tautology checking
    - just check if BDD is identical to function **1**
  - Identity checking
  - Satisfiability
    - look for a path from root to leaf
- All while having a compact representation
  - Use small memory footprint

ROBDD – Sharing

- We already share subtrees within a ROBDD
  - We can share also among multiple ROBDDs!



Logic Operations with ROBDD

- Problem: given two functions G and H, represented by their ROBDDs, compute the ROBDD of a function of (G,H)
- ite operator:
  - ite(f, g, h)
  - If (f) then (g) else (h)
- Recursive paradigm
  - Exploit the generalized expansion of G and H
  - ite (f, g, h) = ite(x, ite(f_x, g_x, h_x), ite($f_{x'}$, $g_{x'}$, $h_{x'}$))

Recursive operation of ITE

Ite(f,g,h)
$= f\,g + f'\,h$
$= v\,(f\,g + f'\,h)_v + v'\,(f\,g + f'\,h)_{v'}$
$= v\,(f_v\,g_v + f'_v\,h_v) + v'\,(f_{v'}\,g_{v'} + f'_{v'}\,h_{v'})$
$= ite(v, ite(f_v, g_v, h_v), ite(f_{v'}, g_{v'}, h_{v'}))$

- Let v be the top-most variable of BDDs f, g, h

Example

- Apply AND to two ROBDDs: f, g
  - fg = ite(f, g, 0)
- Apply OR to two ROBDDs: f, g
  - f+g = ite(f, 1, g)
- Similar for other Boolean operators

Boolean Operators

Operator	Equivalent *ite* form
0	0
$f \cdot g$	$ite(f, g, 0)$
$f \cdot g'$	$ite(f, g', 0)$
f	f
$f'g$	$ite(f, 0, g)$
g	g
$f \oplus g$	$ite(f, g', g)$
$f + g$	$ite(f, 1, g)$
$(f + g)'$	$ite(f, 0, g')$
$f \overline{\oplus} g$	$ite(f, g, g')$
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

Example

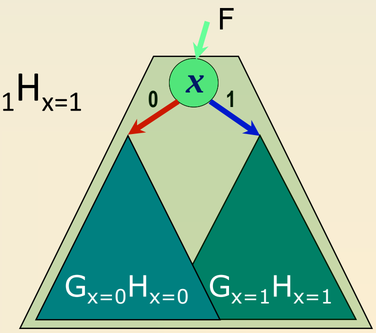
- Compute AND of two ROBDDs
- Terminal cases:
  - AND (0,H) = 0
  - AND (1,H) = H
  - AND (G,0) = 0
  - AND (G,1) = G

Recursive Step

- $G(x,\ldots) = x' \, G_{x=0} + x \, G_{x=1}$
- $H(x,\ldots) = x' \, H_{x=0} + x \, H_{x=1}$

- $F = GH = x' \, G_{x=0} \, H_{x=0} + x \, G_{x=1} H_{x=1}$

  Now we have reduced the problem to computing 2 ANDs of smaller functions



One Last Problem

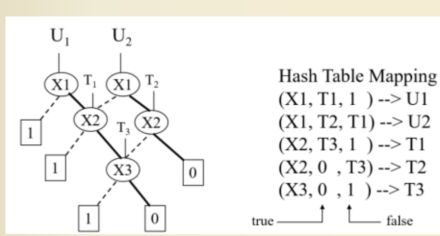
- Suppose, we have computed
  - $G_{x=0} \, H_{x=0}$ and $G_{x=1} \, H_{x=1}$
- We need to construct a new node,
  - label: x
  - 0-cofactor($F_{x=0}$): ROBDD of $G_{x=0} \, H_{x=0}$
  - 1-cofactor($F_{x=1}$): ROBDD of $G_{x=1} \, H_{x=1}$
- BUT, first we need to make sure that we don't violate the reduction rules!

The Unique Table

- To obey reduction rule #1:
  - if $F_{x=0}$ == $F_{x=1}$, the result if just $F_{x=0}$
- To obey reduction rule #2:
  - We keep a unique table of all the BDD nodes and check first if there is already a node
  - $(x, F_{x=0}, F_{x=1})$
- Otherwise, we build the new node
  - and add it to the unique table

The Unique Table

- Unique table : hash table mapping (Xi, 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
  - canonical form : pointer equality determines function equality



Putting All Together

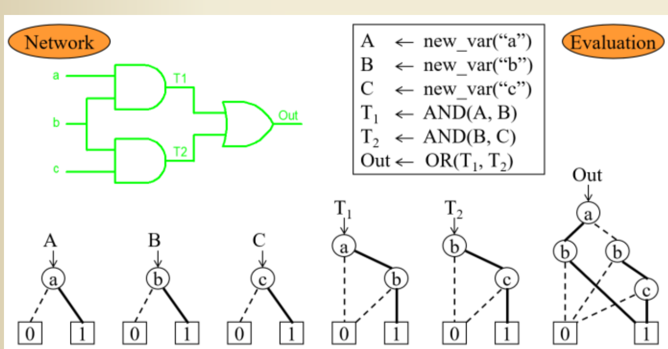
```
AND(G,H) {
        if (G==0) || (H==0) return 0;
        if (G==1) return H;
        if (H==1) return G;
        cmp = computed_table_lookup(G,H);
        if (cmp != NULL) return cmp;

        x = top_variable(G,H);
        G1 = G.then; H1 = H.then;
        G0 = G.else; H0 = H.else;
        F0 = AND(G0,H0);
        F1 = AND(G1,H1);
        if (F0 == F1) return F0;
        F = find_or_add_unique_table(x,F0,F1);
        computed_table_insert(G,H,F);
        return F;
}
```

2024/3/21 Andy Yu-Guang Chen 43

Generating ROBDD from Network

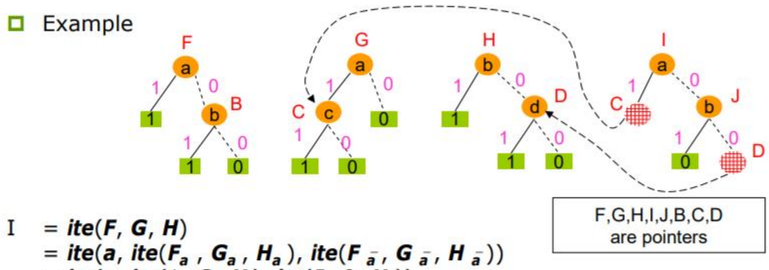
- Task : Represent output functions of gate network as ROBDDs



2024/3/21 Andy Yu-Guang Chen 44

Example

- Example

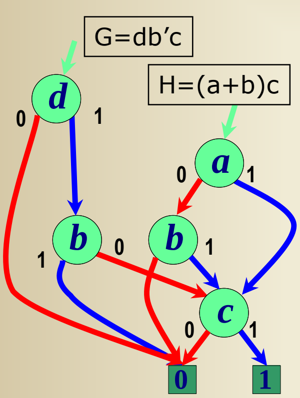


I $= \boldsymbol{ite}(\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H})$
$= \boldsymbol{ite}(\boldsymbol{a}, \boldsymbol{ite}(\boldsymbol{F_a}, \boldsymbol{G_a}, \boldsymbol{H_a}), \boldsymbol{ite}(\boldsymbol{F_{\bar a}}, \boldsymbol{G_{\bar a}}, \boldsymbol{H_{\bar a}}))$
$= \boldsymbol{ite}(\boldsymbol{a}, \boldsymbol{ite}(1, \boldsymbol{C}, \boldsymbol{H}), \boldsymbol{ite}(\boldsymbol{B}, 0, \boldsymbol{H}))$
$= \boldsymbol{ite}(\boldsymbol{a}, \boldsymbol{C}, \boldsymbol{ite}(\boldsymbol{b}, \boldsymbol{ite}(\boldsymbol{B_b}, 0_b, \boldsymbol{H_b}), \boldsymbol{ite}(\boldsymbol{B_{\bar b}}, 0_{\bar b}, \boldsymbol{H_{\bar b}}))$
$= \boldsymbol{ite}(\boldsymbol{a}, \boldsymbol{C}, \boldsymbol{ite}(\boldsymbol{b}, \boldsymbol{ite}(1, 0, 1), \boldsymbol{ite}(0, 0, \boldsymbol{D})))$
$= \boldsymbol{ite}(\boldsymbol{a}, \boldsymbol{C}, \boldsymbol{ite}(\boldsymbol{b}, 0, \boldsymbol{D}))$
$= \boldsymbol{ite}(\boldsymbol{a}, \boldsymbol{C}, \boldsymbol{J})$

Check: F = a + b
G = ac
H = b + d
$\boldsymbol{ite}(\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}) = (a + b)(ac) + a'b'(b + d) = ac + a'b'd$

Logic Operations – Example

Order:d < a < b < c



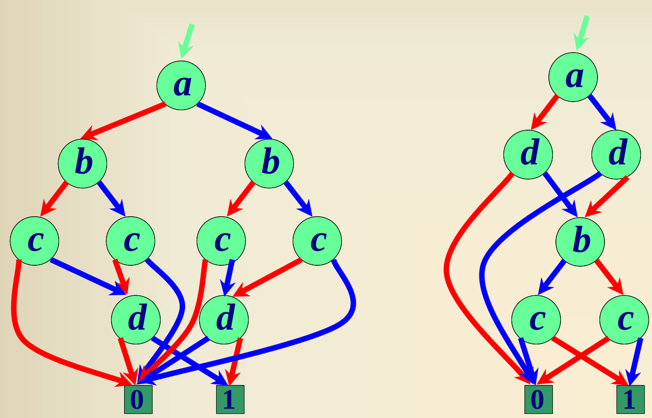
Split Variable	G cof	H cof	AND(G,H)
d=0	0	(a+b)c	
d=1	b’c	(a+b)c	
d=1,a=0			

Logic Operations – Summary

- Recursive routines – traverse the DAGs depth first
- Two tables:
  - Unique table – hash table with and entry for each BDD node
  - Computed table – store previously computed partial results
- To perform other operations, just change the terminal cases

The Importance of Variable Order

$$F = (a \oplus d)(b \oplus c)$$



Ordering Results

Function type	*Best order*	*Worst order*
addition	linear	exponential
symmetric	linear	quadratic
multiplication	exponential	exponential

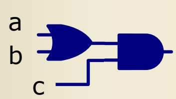
- In practice:
  - Many common functions have reasonable size
  - Can build ROBDDs with millions of nodes
  - Algorithms to find good variables ordering

Variable Ordering Algorithms

- Problem: given a function F, find the variable order that minimizes the size of its ROBBDs
- Answer: problem is intractable
- Two heuristics
  - Static variable ordering (1988)
  - Dynamic variable ordering (1993)

Static Variable Ordering

- Variables are ordered based on the network topology
  - How: put at the bottom the variables that are closer to circuit's outputs
  - Why: because those variables only affect a small part of the circuit



good order: a < b < c

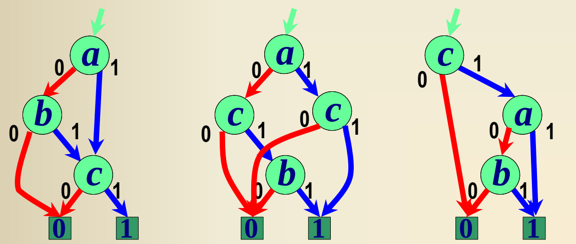
  - Disclaimer: it's a heuristic, results are not guaranteed

Dynamic Variable Ordering

- Changes the variable order on-the-fly whenever ROBDDs become too big
- How: trial and error – SIFTING ALGORITHM
  1. Choose a variable
  2. Move it in all possible positions of the variable order
  3. Pick the position that leaves you with the smallest ROBDDs
  4. Choose another variable …

Dynamic Variable Ordering

- Tiny example: F=(a+b)c
  - we want to find the optimal position for variable c



initial order: a < b < c

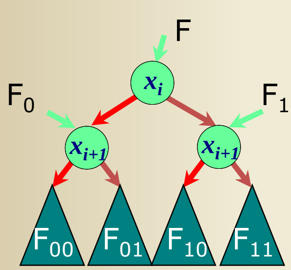
Swap (b, c): a < c < b

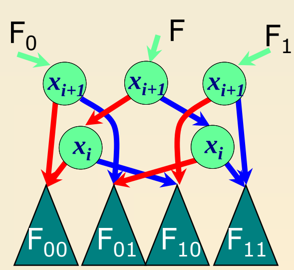
Swap (a, c): c < a < b

Final order: c<a<b

Variable Swapping

$$ITE(x_i, F_1, F_0) =$$
$$= ITE(x_i, ITE(x_{i+1}, F_{11}, F_{10}), ITE(x_{i+1}, F_{01}, F_{00}))$$
$$= ITE(x_{i+1}, ITE(x_i, F_{11}, F_{01}), ITE(x_i, F_{10}, F_{00}))$$



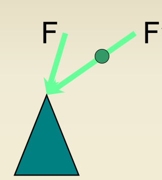


Dynamic Variable Ordering

- Key idea: swapping two variables can be done locally
  - Efficient:
    - Can be done just by sweeping the unique table
  - Robust:
    - Works well on many more circuits
  - Warning:
    - The technique is still non optimal
    - At convergence, you most probably have found only a local minimum

Improvements of BDDs

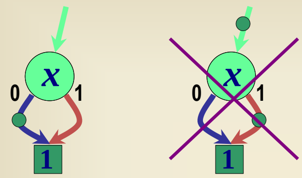
- Complement edges (1990)
  - Creates more opportunities for sharing
    → fewer nodes
  - For every pair (F,F'), we
    - only construct the ROBDD for F
    - F' is given by using a complement edge to F
  - Which do you pick?
    - THEN edge can never be complemented
    - Only constant value



1

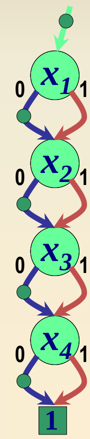
Complement Edges

- F = x



$$F = x_1 \oplus x_2 \oplus x_3 \oplus x_4$$

- Still canonical



Summary

- BDDs
  - Very efficient data structure
  - Efficient manipulation routines
  - A few important functions don’t come out well
  - Variable order can have a high impact on size
- Application in many areas of CAD
  - Hardware verification
  - Logic synthesis

Q&A

Andy, Yu-Guang Chen
Assistant Professor, Department of EE, NCU
Email: andyygchen@ee.ncu.edu.tw
FB: Yu-Guang Chen
IG: ncu.eda.andy
Google account: andyygchen.ncu