Logic Representations

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 mapping.

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

- For each component f_i, $i = 1,2, \ldots, m$, define

ON_SET:the set of input values x such that $f_i(x) = 1$

OFF_SET:the set of input values x such that $f_i(x) = 0$

DC_SET:the set of input values x such that $f_i(x) = 2$

- Completely specified function : DC_SET = ϕ

  Incompletely specified function : DC_SET$\neq \phi$

- m=1 => a single output function

  m>1 => a multiple output function

Representations

1. Truth Table

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

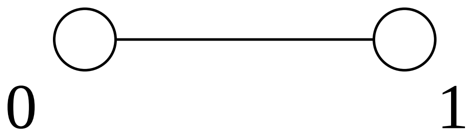
- Multiple output function
- Sum : on-set = {(0 0 1), (0 1 0), (1 0 0), (1 1 1)}

  off-set= {(0 0 0), (0 1 1), (1 0 1), (1 1 0)}
- Completely specified function

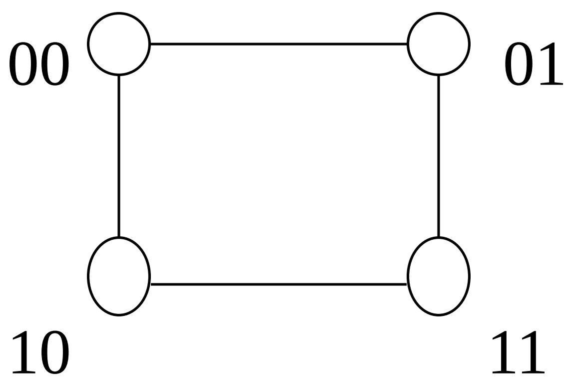
Representations

2. Geometrical representation

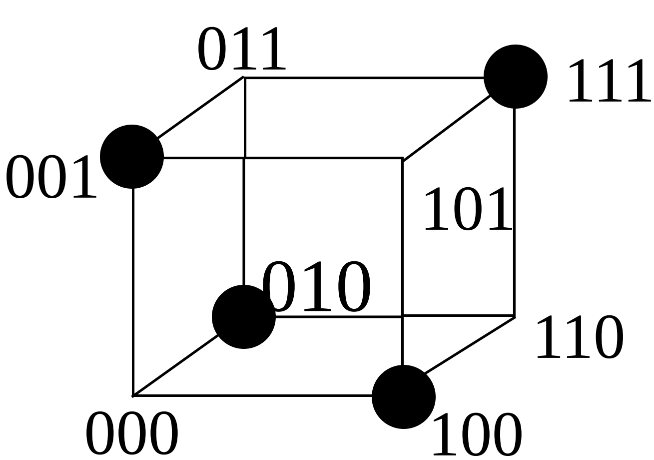
1 variable



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)}

Representations

3. Algebraic representations

Canonical sum of product (list of minterms)

$$C_{out} = x'yC_{in} + xy'C_{in} + xyC_{in}' + xyC_{in}$$

Reduced sum of product (two-level)

$$C_{out} = yC_{in} + xC_{in} + xy$$
$$= yC_{in} + xC_{in} + xyC_{in}'$$

- List of cubes (Sum of Products (SOP), Disjunctive Normal Form (DNF))
- List of conjuncts (Product of Sums (POS), Conjunctive Normal Form (CNF))

Multi-level representation

$$C_{out} = C_{in} (x + y) + xy$$

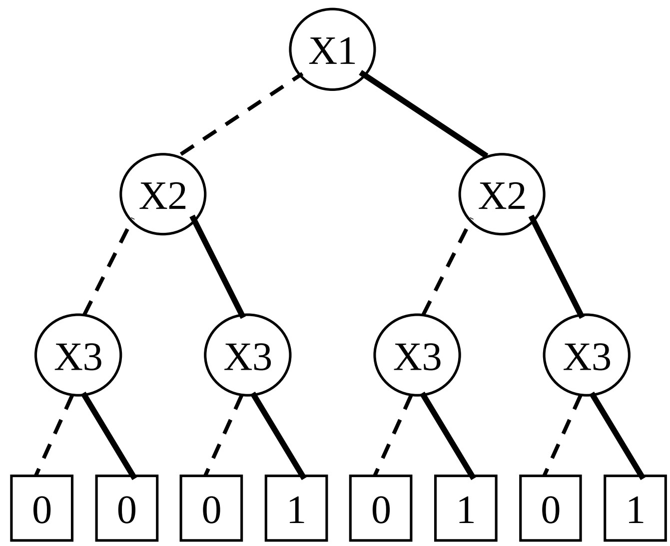
4. Reduced Ordered Binary Decision Diagrams

Binary Decision Diagram

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

Decision Tree

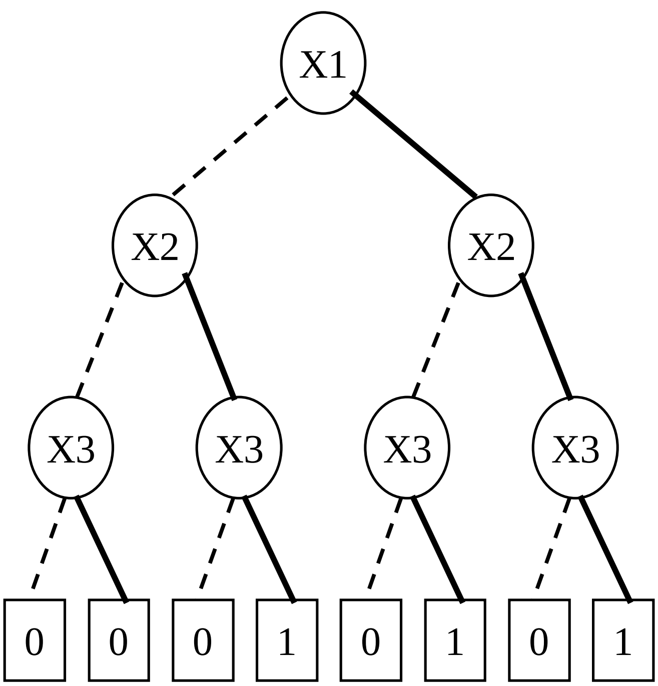


- Vertex represents decision
- Follow dashed line for value 0
- Follow solid line for value 1
- Function value determined by leaf value

Reduced Ordered Binary Decision Diagram (ROBDD)

Binary decision graph:

$$f = x_2x_3 + x_1x_3$$



- Terminal node:

• attribute

value (v) = 0

value (v) = 1

- Nonterminal node:

• index (v) = i, i= 1~3

• two children

low (v)

high (v)

- Evaluate an input vector

ROBDD

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 value (v) is 1, then $F_v = 1$

b) if value (v) is 0, then $F_v = 0$

(2) If F is a nonterminal node (with index(v) = i)

$$F_v(x_i , ...x_n) = x_i' F_{low(v)} (x_{i+1} , ...x_n) + x_i F_{high(v)} (x_{i+1} , ...x_n)$$

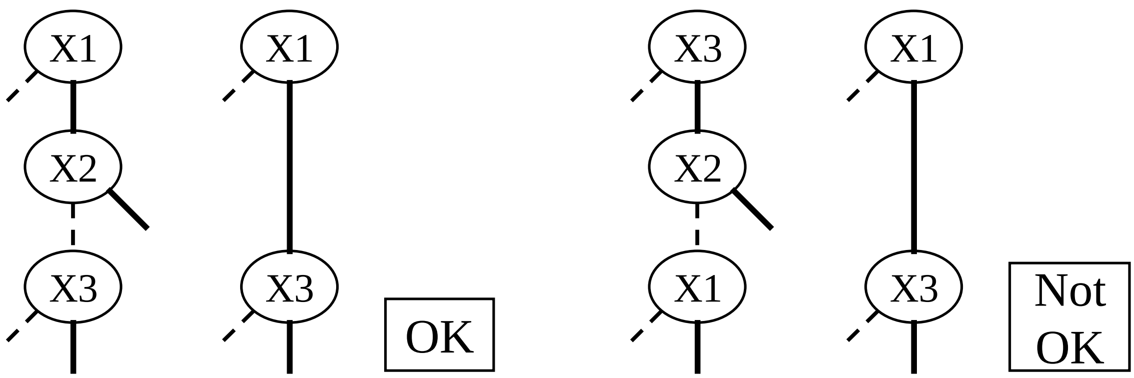
Shannon expansion

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

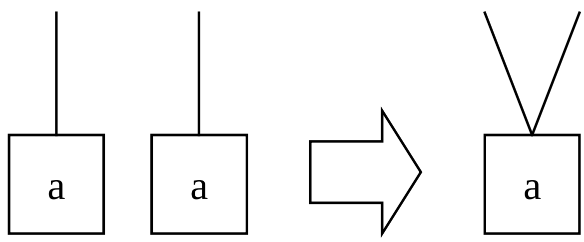
Reduced OBDD

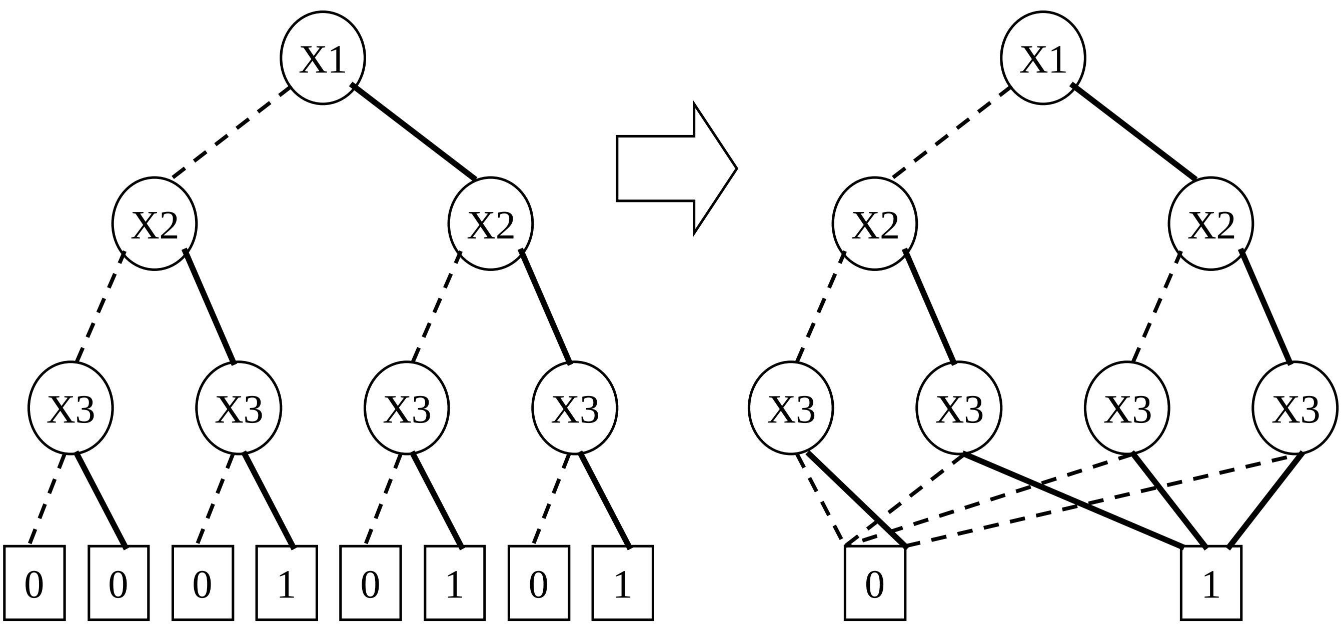
Reduced OBDD :

1. No distinct vertices v and w such that subgraphs rooted by v and w are isomorphism.
2. No vertex v with low (v) = high (v)

Reduction Rule #1

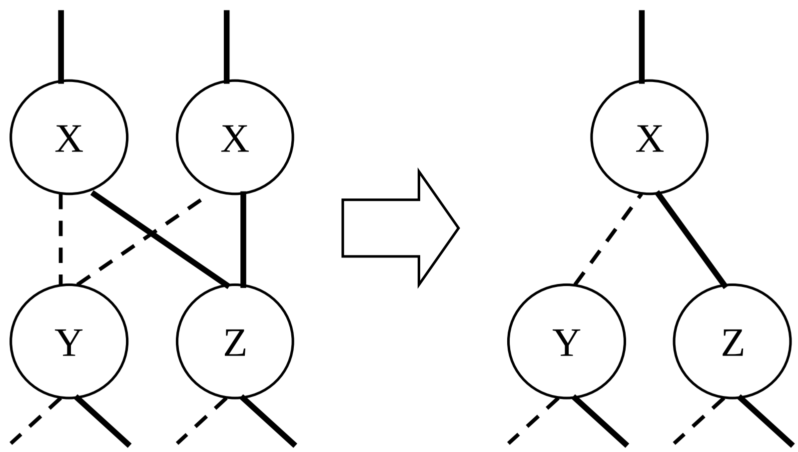
- Merge equivalent leaves

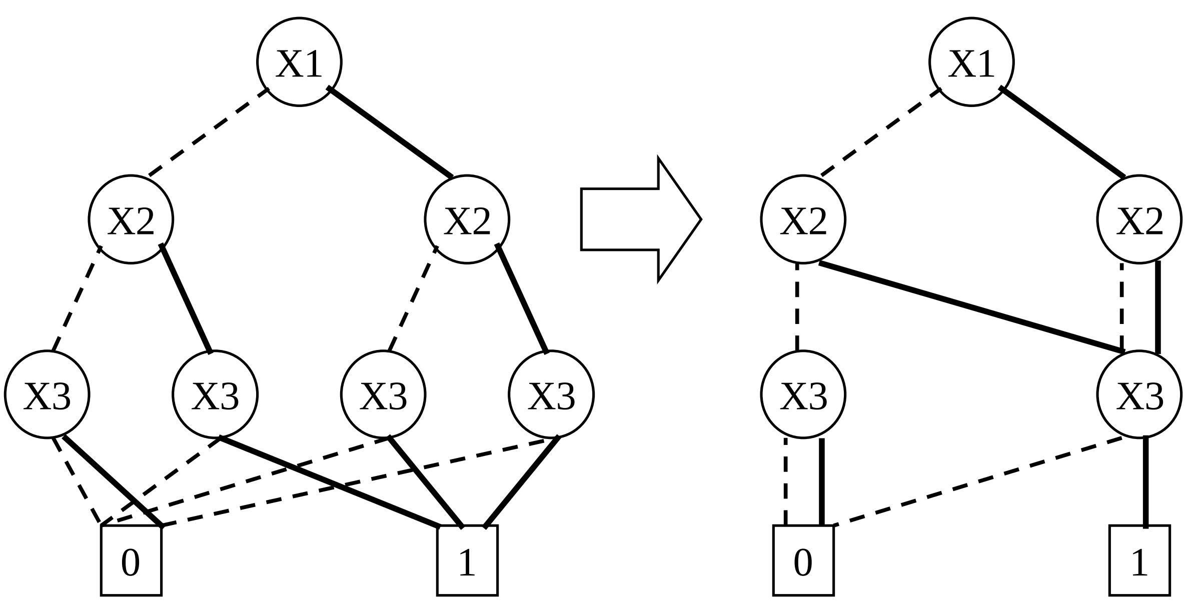




Reduction Rule #2

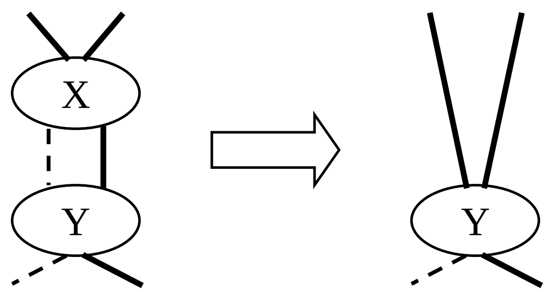
- Merge isomorphic nodes

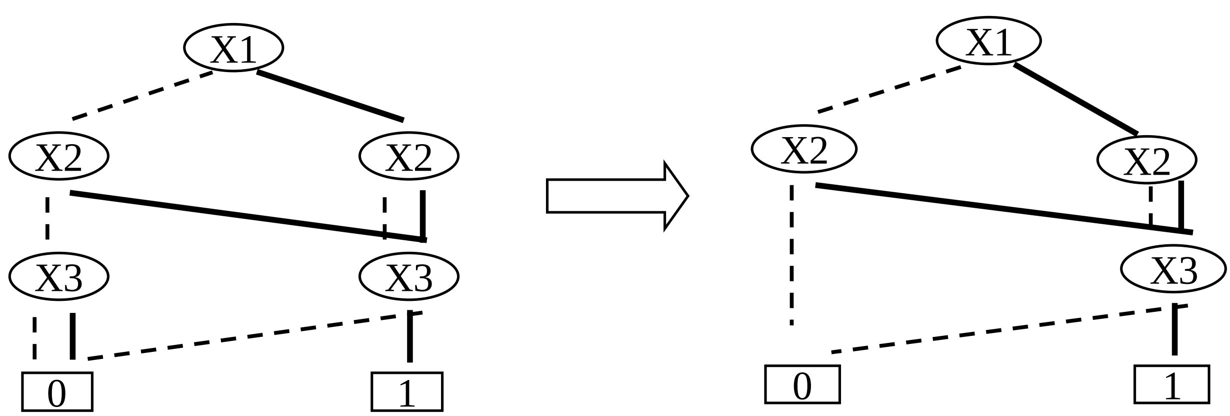




Reduction Rule #3

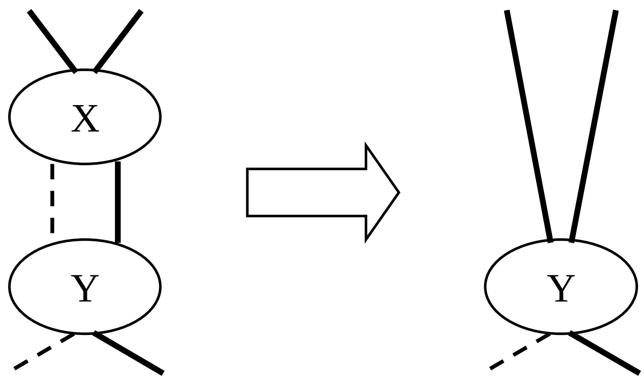
- Eliminate redundant node

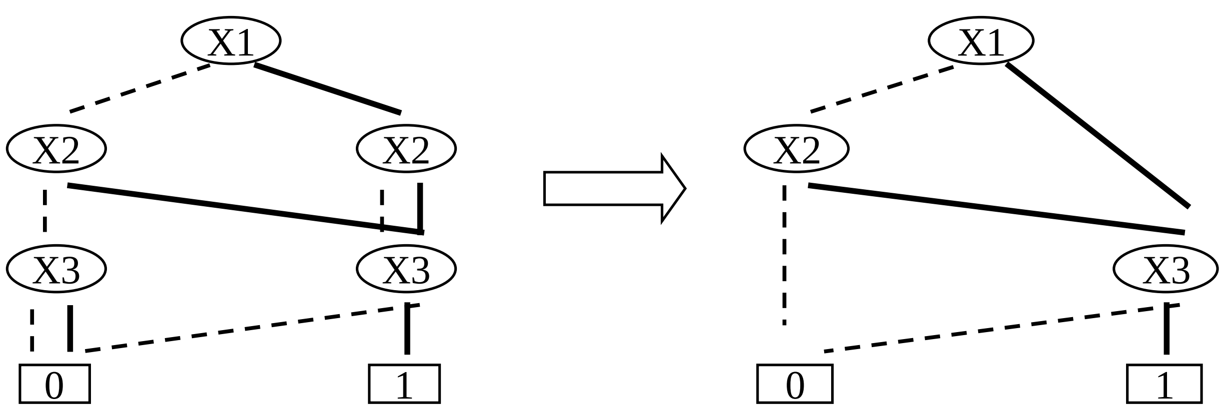




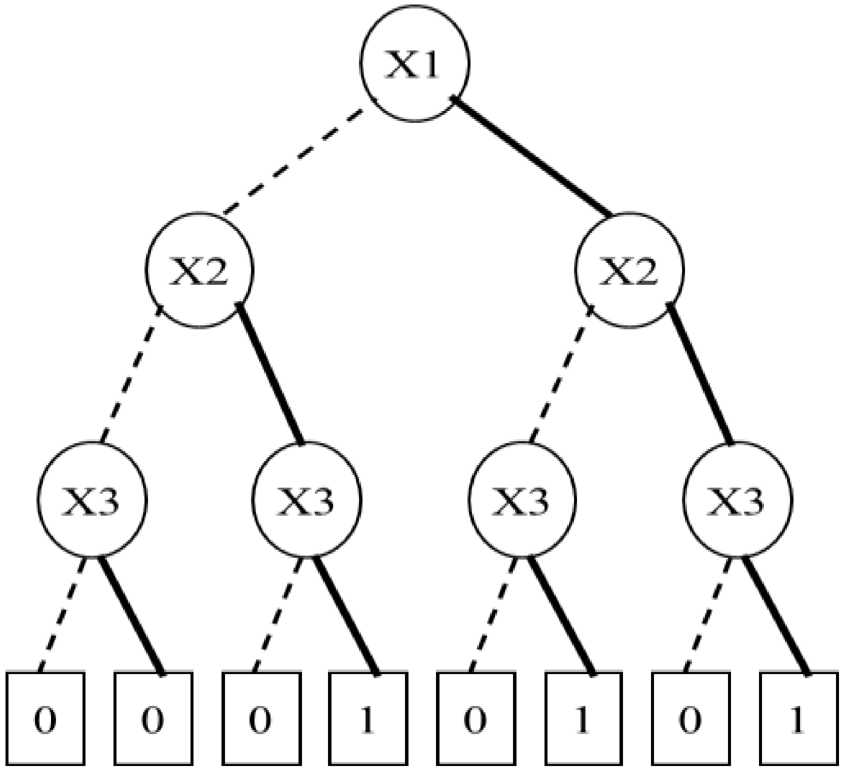
Reduction Rule #3

- Eliminate redundant node

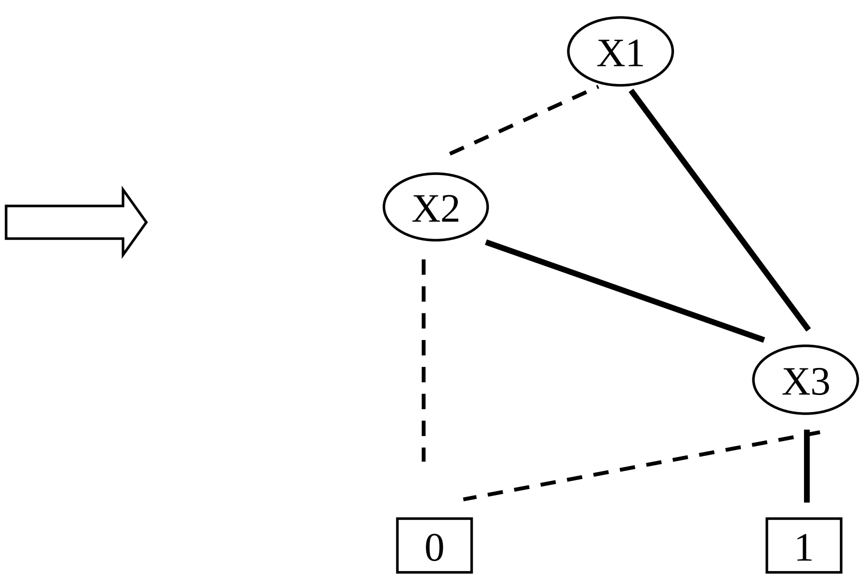




ROBDD



After reduce

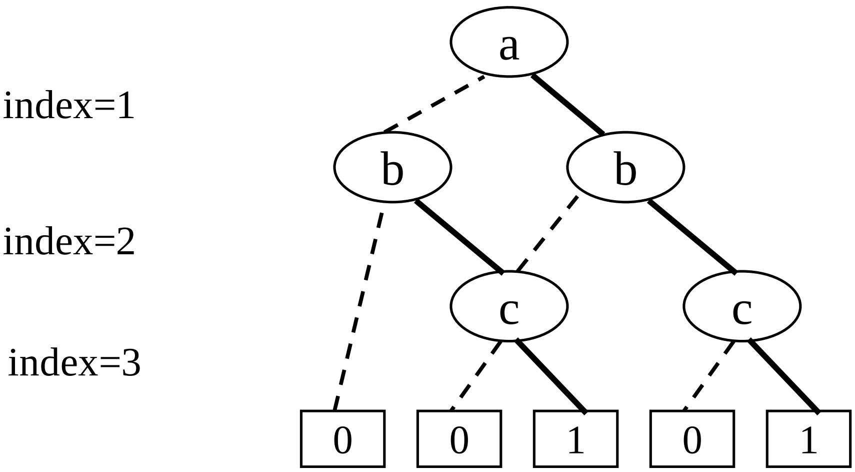


Implementation of Reduce

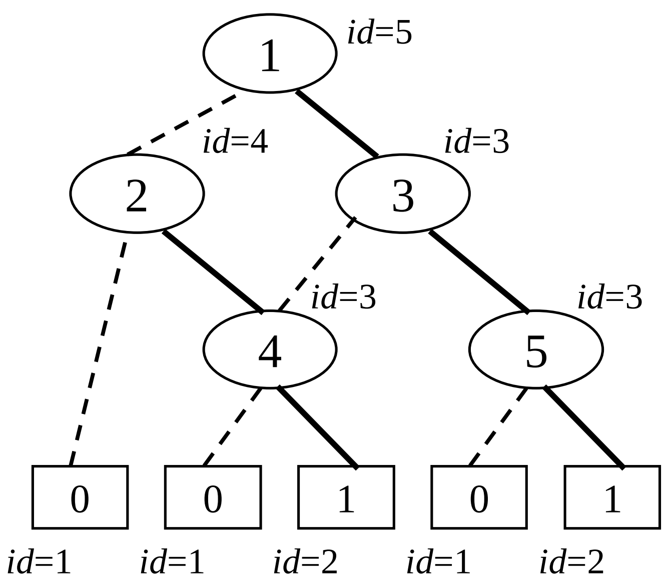
- Visit OBDD bottom up and label each vertex with an identifier
- Redundancy
  - if id(low(v)) = id(high(v)), then vertex v is redundant

    $\Rightarrow$ set id(v) = id(low(v))
  - if index (v) = index(u) and id(low(v)) = id(low(u)) and id(high(v)) = id(high(u)), then set id(v) = id(u)
- A different identifier is given to each vertex at level i
- An ROBDD is identified by a subset of vertices with different identifiers

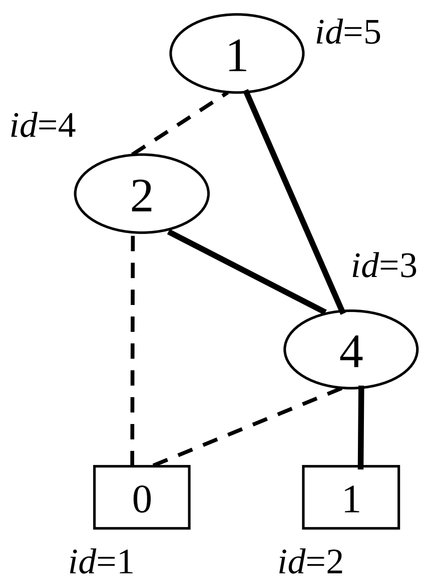
Reduce


index=1
index=2
index=3
a
b
b
c
c
0
0
1
0
1

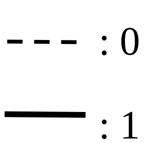
(a)


1
id=5
2
id=4
3
id=3
4
id=3
5
id=3
0
0
1
0
1
id=1
id=1
id=2
id=1
id=2

(b)


1
id=5
id=4
2
id=3
4
0
1
id=1
id=2

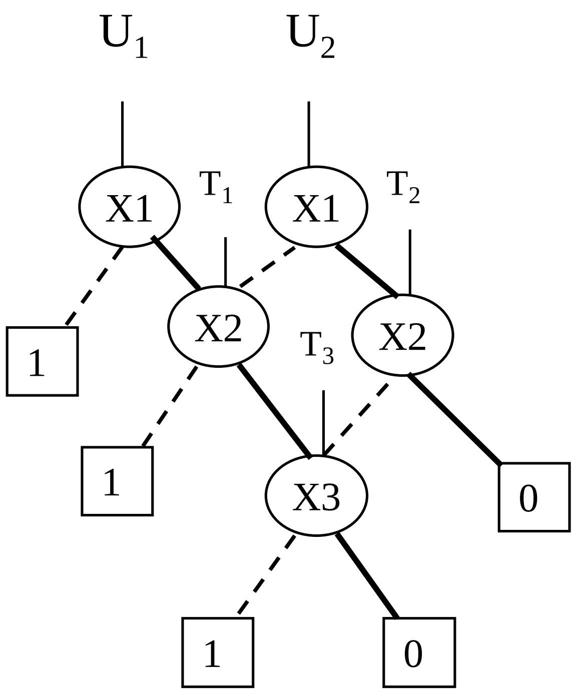
(c)


: 0
: 1

Construct ROBDD Using Hash Table

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

The Unique Table – Hash Table

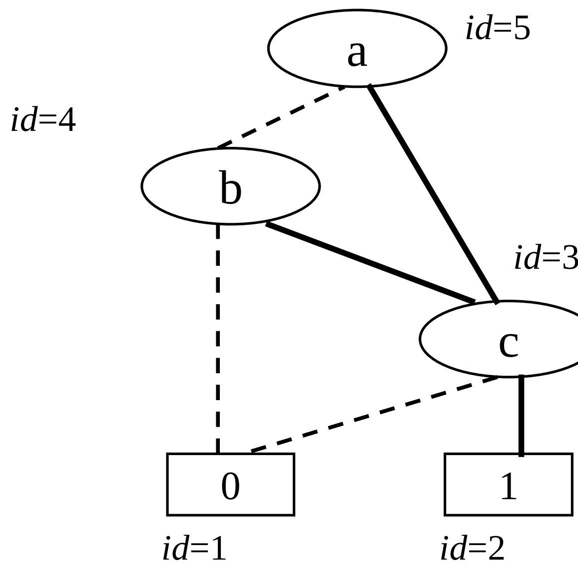


Hash Table Mapping
(X1, T1, 1) --> U1
(X1, T2, T1) --> U2
(X2, T3, 1) --> T1
(X2, 0 , T3) --> T2
(X3, 0 , 1) --> T3

true false

- Unique table : hash table mapping (Xi, G, H) into a node in the DAG
  - before adding a node to the DAG, check if it already exists
  - avoids creating two nodes with the same function
  - constructed bottom up
  - terminated when root is reached
  - canonical form : pointer equality determines function equality

ROBDD Using Unique Table



f = (a+b) c

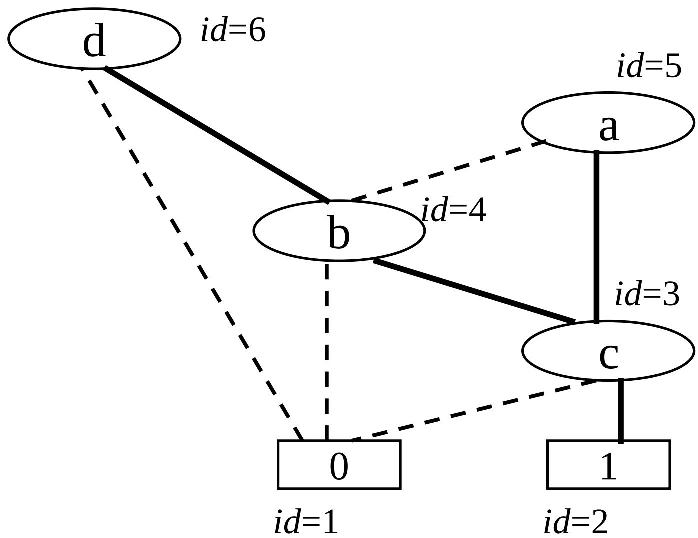
variable order (a, b, c)

Unique table

Key

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

Multi-Rooted ROBDD



f = (a+b) c

g = b c d

variable order (d, a, b, c)

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

Ordering Effects

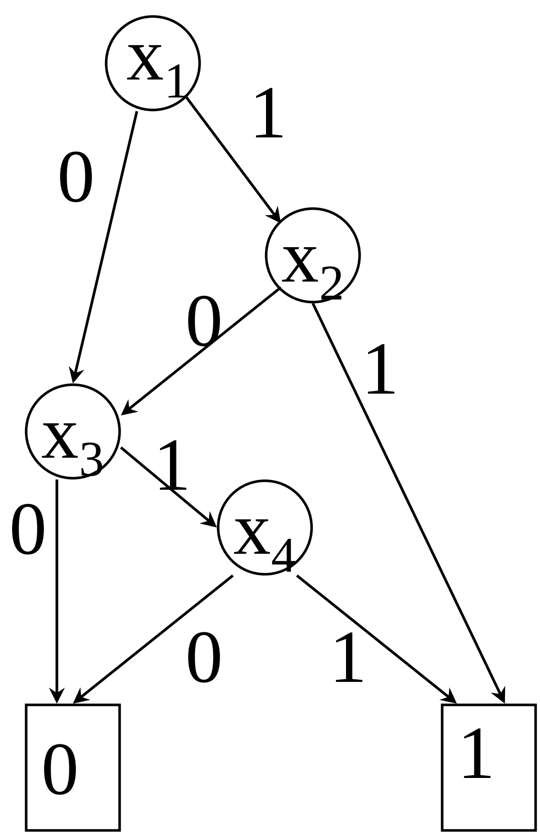
- Ordered BDD: if x < y, then all nodes representing x precede all nodes representing y
- Given an ordering of variables, reduced OBDD is canonical

Example of a Good Ordering

- The size of OBDD depends on the ordering of variables

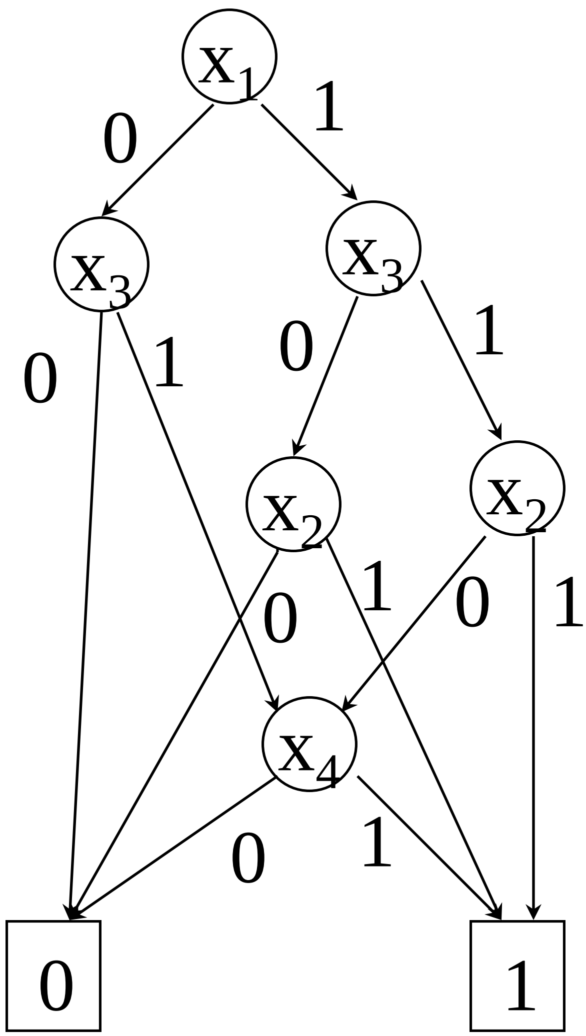
ex : $x_1x_2 + x_3x_4$

$x_1 < x_2 < x_3 < x_4$



Example of a Less Good Ordering

- For a good ordering, the size of an OBDD remains reasonably small
- Multiplier is an exception



$x_1 < x_3 < x_2 < x_4$

Which Ordering is Better?

- Let x3 x2 x1 x0 and y3 y2 y1 y0 be two binary numbers and carry-out3 the carry out of the most significant bit.

- Two ordering :

$$x3 < x2 < x1 < x0 < y3 < y2 < y1 < y0$$

$$x3 < y3 < x2 < y2 < x1 < y1 < x0 < y0$$

- Which ordering will result in smaller size and why?

Sample Function Classes

Function Class	Best	Worst	Ordering Sensitivity
ALU (Add/Sub)	Linear	Exponential	High
Symmetric	Linear	Quadratic	Medium
Multipication	Exponential	Exponential	Low

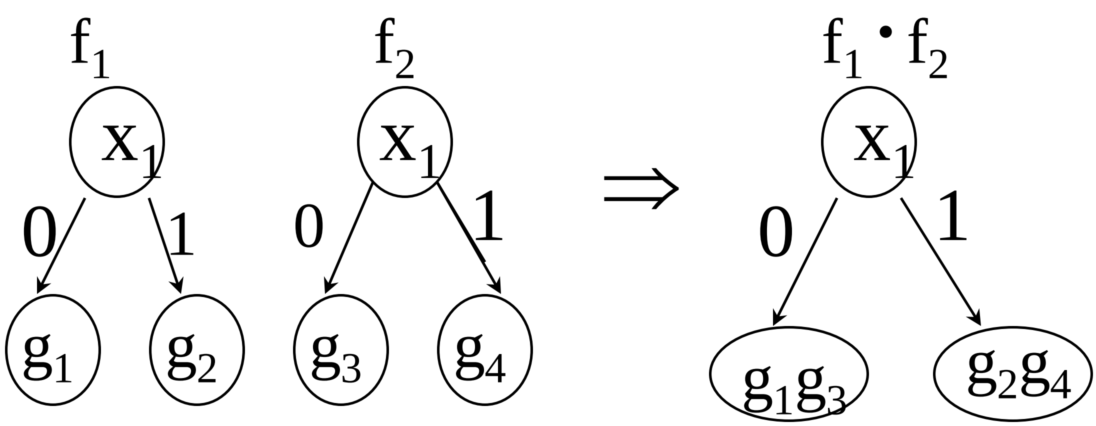
- General Experience
  - Many tasks have reasonable ROBDD representations
  - Algorithms remain practical for up to 100,000 vertex ROBDD
  - Heuristic ordering methods generally satisfactory

Operations on ROBDD

Apply f_1 <op> f_2

(f_1, f_2 must have the same ordering of variables, then operations can be operated on f_1, f_2)

- $f_1 \bullet f_2 = (x_1'g_1 + x_1g_2)(x_1'g_3 + x_1g_4)$
  $= x_1'g_1g_3 + x_1g_2g_4$
  $= x_1'(g_1g_3) + x_1(g_2g_4)$



- $f_1 + f_2$
- $f_1 \oplus f_2$
- etc.

ROBDD

- complement : $f \oplus 1$
- $f1 \rightarrow f2$: $f1' + f1f2$ (implication)
- time complexity

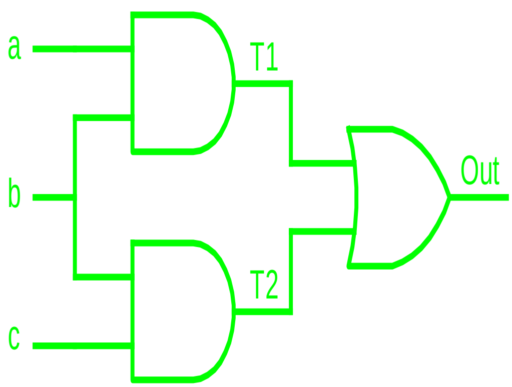
Reduced $O(|G| \log |G|)$

Apply $O(|G_1||G_2|)$

Generating ROBDD from Network

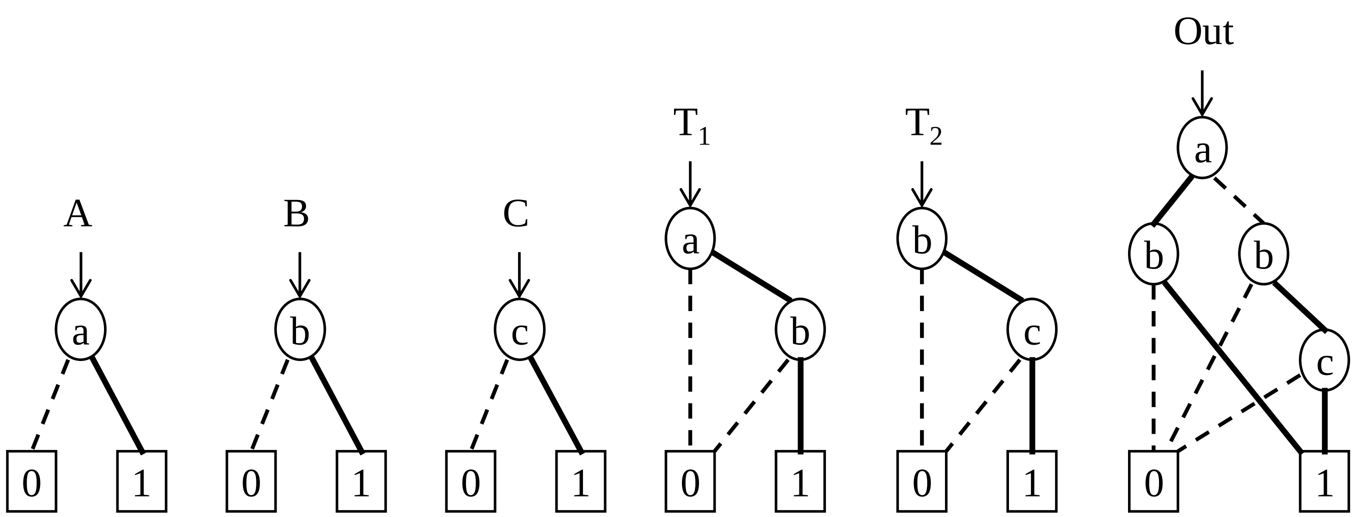
- Represent output functions of gate network as ROBDDs

Network



Evaluation

$A \leftarrow$ new_var("a")
$B \leftarrow$ new_var("b")
$C \leftarrow$ new_var("c")
$T_1 \leftarrow$ AND(A, B)
$T_2 \leftarrow$ AND(B, C)
Out $\leftarrow$ OR(T_1, T_2)



Property of ROBDD

(1) Commonly encountered functions have reasonable representations.(except multiplier)
(2) Complementation will not blow up the representation.
(3) Canonical form so that equivalence, satisfiability checking can be done easily.
(4) Multi-level representation

Achille’s heel function:

$F = x_1x_2x_3 + x_4x_5x_6 + \ldots\ldots + x_{3n-2}x_{3n-1}x_{3n}$

F’ has 3^n terms

5. And-Inverter Graphs (AIGs)

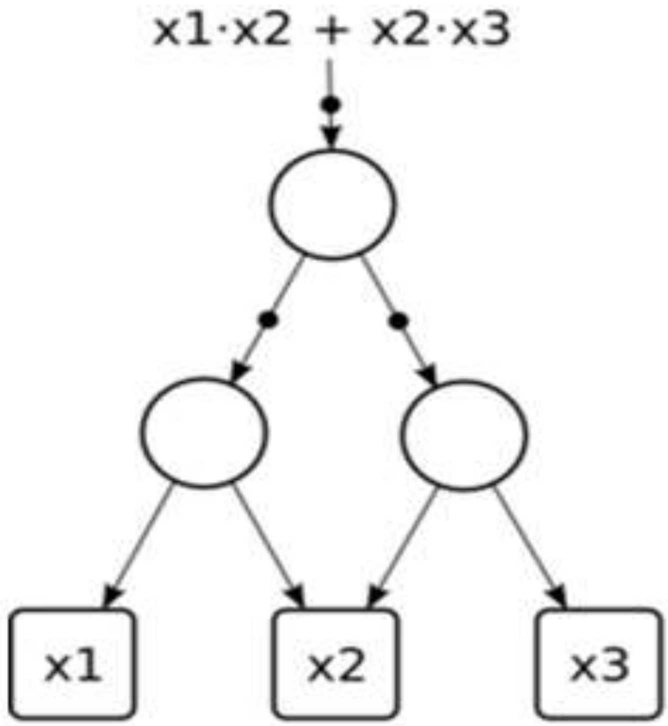
And-Inverter Graph

5. And-Inverter Graph (AIG)
   - Simple structure
   - And-Gates as nodes (shown as circles) with two inputs as edges (shown as arrows)
   - Inverter edges marked with a dot
   - Used in ABC

Example

Ex: $y = f(x_1,x_2,x_3) = ((x_1 \cdot x_2)' \cdot (x_2 \cdot x_3)')'$

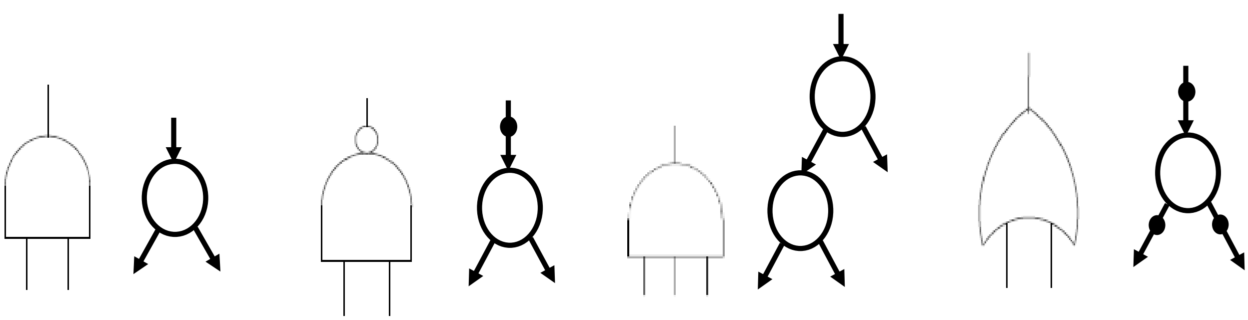
$= (x_1 \cdot x_2) + (x_2 \cdot x_3)$



AIG Construction

- Start from SOP representation
- Convert to AIG using DeMorgan's law

$$x_1 + x_2 = (x_1 + x_2)'' = (x_1' \cdot x_2')'$$

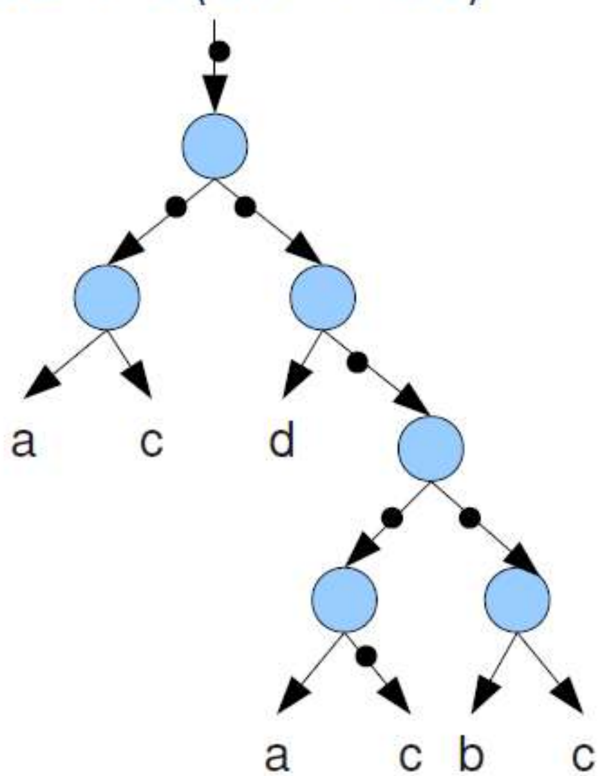


2-input and   2-input nand   3-input and   2-input or

AIG Attributes

- Size is the number of AND nodes in it
- Logic level is the number of AND-gates on the longest path from the primary inputs (PIs) to the primary outputs (POs)
- The inverters are ignored

$f(a,b,c,d) =$
$ac + d(ac' + bc)$



6 nodes, 4 levels

AIG Canonicity

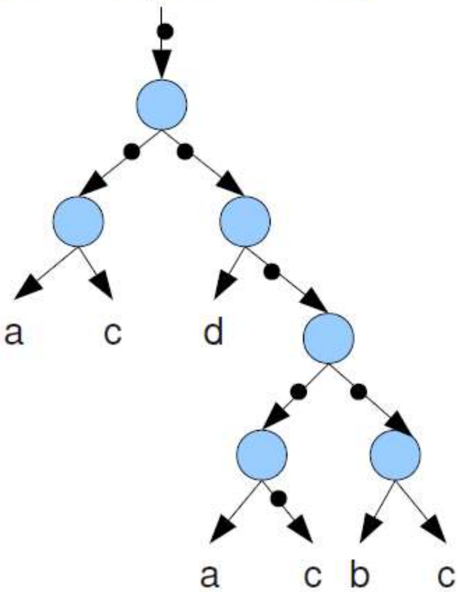
- AIGs are not canonical
  - ROBDDs are canonical
  - Same function represented by two functionally equivalent AIGs with different structures
  - Different structures can still be optimal with different objectives

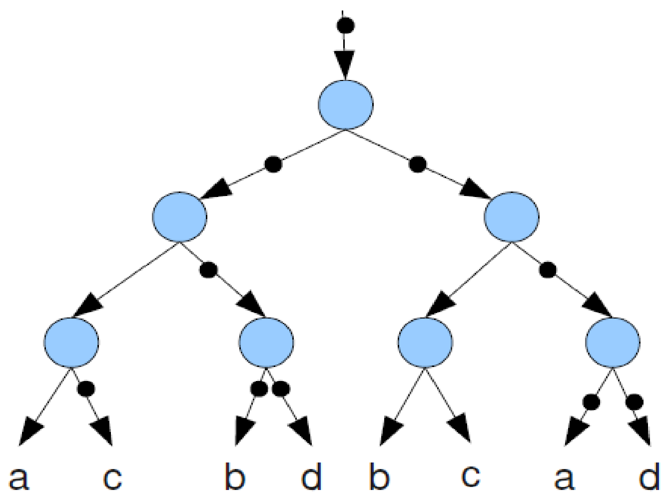
AIG Canonicity

$f(a,b,c,d) =$

$ac + d(ac' + bc)$

$= \quad ac'(b'd')' + bc(a'd')'$





6 nodes, 4 levels
=> area optimal

7 nodes, 3 levels
=> speed (delay) optimal

6. Reed-Muller Form

Representation

Reed-Muller Expression (with AND, XOR and 1)

Algebraic Normal Form (ANF)

- $f(x_1, x_2, x_3, \dots, x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n \oplus a_{1,2} x_1 x_2 \oplus a_{1,3} x_1 x_3 \oplus \cdots \oplus a_{n-1,n} x_{n-1,n} \oplus \cdots \oplus a_{1,2,\dots,n} x_1 x_2 \dots x_n$

  where $a_i = 0 \; or \; 1$
- Use un-complemented literals only
- The above is called the Positive Polarity RM (PPRM), which is canonical (unique) for the given function
- $a \oplus b = b \oplus a, \;\; a \oplus 1 = a', \;\; (a \oplus b) \oplus c = a \oplus (b \oplus c), \;\; c(a \oplus b) = ac \oplus bc$
- $a + b = a \oplus b$ when a, b are exclusive

Example

$$
\begin{aligned}
f(x_1, x_2) &= f(0, 0)x_1'x_2' + f(0, 1)x_1'x_2 + f(1, 0)x_1x_2' \\
&\quad + f(1, 1)x_1x_2 \\
&= f(0, 0)x_1'x_2' \oplus f(0, 1)x_1'x_2 \oplus f(1, 0)x_1x_2' \\
&\quad \oplus f(1, 1)x_1x_2 \\
&= f(0, 0)(x_1\oplus 1)(x_2\oplus 1) \oplus \\
&\quad f(0, 1)(x_1\oplus 1)x_2 \oplus \\
&\quad f(1, 0)x_1(x_2\oplus 1) \oplus \\
&\quad f(1, 1)x_1x_2 \\
&= f(0, 0)(x_1x_2\oplus x_1\oplus x_2\oplus 1) \oplus \\
&\quad f(0, 1)(x_1x_2\oplus x_2) \oplus \\
&\quad f(1, 0)(x_1x_2\oplus x_1) \oplus \\
&\quad f(1, 1)x_1x_2 \\
&= f(0, 0) \oplus \\
&\quad (f(0,0)\oplus f(1,0))x_1 \oplus \\
&\quad (f(0,0)\oplus f(0,1))x_2 \oplus \\
&\quad (f(0,0)\oplus f(0,1)\oplus f(1,0) \oplus f(1,1))x_1x_2
\end{aligned}
$$

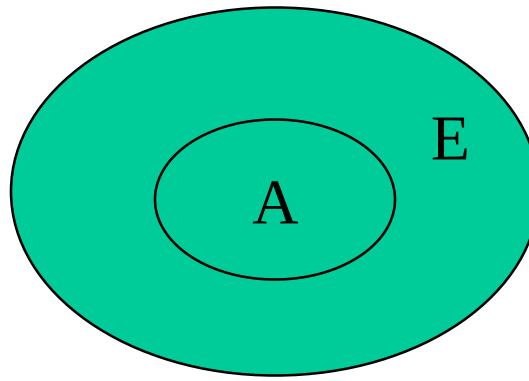
Characteristic Functions

Characteristic Function

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

The characteristic function A is the function

$$X_A : E \to \{ 0,1 \}$$
$$X_A(x) = 1 \; if \; x \in A$$
$$X_A(x) = 0 \; if \; x \notin A$$



Ex:

$$E = \{ 1,2,3,4 \}$$
$$A = \{ 1,2 \}$$
$$X_{A(1)} = 1$$
$$X_A(3) = 0$$

Characteristic Function

Given a Boolean function

$$f\ :\ Bn\ \rightarrow\ Bm$$

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 $(x, y) \in F$

Characteristic Function

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

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$F_y(x_1, x_2, y) \;=$

x_1	x_2	y	F	
0	0	0	1	v
0	0	1	0	
0	1	0	0	
0	1	1	1	v
1	0	0	0	
1	0	1	1	v
1	1	0	0	
1	1	1	1	v

Operations on Logic Functions

Operation on Logic Function

- Complement
- Intersection
- Union
- Difference
- XOR
- F is a tautology
- Cofactor
- Boolean difference
- Consensus operator
- Smoothing operator

Cofactor

Cofactor operation (restriction)

$f_{\overline{xi}} = f_{xi=0}$ cofactor of f with respect to x_i=0

$f_{xi} = f_{xi=1}$ cofactor of f with respect to x_i=1

Cofactor is a new function independent of x_i

$$f_{xi=0} = f(x_1, x_2, \ldots, x_i = 0, \ldots, x_n)$$
$$f_{xi=1} = f(x_1, x_2, \ldots, x_i = 1, \ldots, x_n)$$

$Ex:$

$$f(x, y, z) = xy + \overline{y}z + \overline{x}\overline{z}$$
$$f_{x=0} = \overline{y}z + \overline{z}$$
$$f_{x=1} = y + \overline{y}z$$

Cofator

Cofactor with respect to any cube

Ex:

$$f(x, y, z, w) = xy + \bar{z}w + \overline{w}\ \overline{x}$$

$$f_{\overline{x}\overline{y}} = f_{x=0, y=0} = \bar{z}w + \overline{w}$$

$$f_{x\bar{y}} = f_{x=1, y=0} = \bar{z}w$$

Shannon Expansion

$$f = \bar{x} \cdot f_{x=0} + x \cdot f_{x=1}$$
$$= \bar{x}_i \cdot \bar{y}_j \cdot f_{\bar{x}_i \bar{y}_j} + x_i \cdot \bar{y}_j \cdot f_{x_i \bar{y}_j} + \bar{x}_i \cdot y_j f_{\bar{x}_i y_j} + x_i \cdot y_j \cdot f_{x_i y_j}$$

$$\text{Ex}: \qquad f(x, y, z, w) = xy + z\overline{w} + \overline{x}\overline{w}$$
$$f_x = y + z\overline{w}$$
$$f_{\bar{x}} = z\overline{w} + \overline{w}$$
$$f = x(y + z\overline{w}) + \bar{x}(z\overline{w} + \overline{w})$$

Boolean Difference

$\frac{\partial f}{\partial x}$ is called Boolean difference of f with respect to x

$$Def\colon \frac{\partial f}{\partial x} = f_x \oplus f_{\overline{x}}$$

f is sensitive to the value of x when $\frac{\partial f}{\partial x}$

Ex:

$$f(x, y, z, w) = xy + z\overline{w} + \overline{w}\ \overline{x}$$

$$\mathrm{f}_{\overline{\mathrm{x}}} = f(x = 0, y, z, w) = z\overline{w} + \overline{w}$$

$$\mathrm{f}_{\mathrm{x}} = f(x = 1, y, z, w) = y + z\overline{w}$$

$$f_{\overline{x}} \oplus f_x = (z\overline{w} + \overline{w}) \oplus (y + z\overline{w})$$

$$= yw$$

Application: Test pattern generation

Consensus Operator

$$Def: \ \forall x(f) = f_x \cdot f_{\overline{x}}$$

$\forall x(f)$ evaluate f to be true for *x=1* and *x=0*

Ex: $$f(x, y, z, w) = xy + z\overline{w} + \overline{w}\ \overline{x}$$

$$\forall x(f) = f_x \cdot f_{\overline{x}} = z\overline{w} + \overline{w}\ y$$

. Universal quantification of function w.r.t. variable x

Smoothing Operator

$$Def: \ \exists x(f) = f_x + f_{\bar{x}}$$

$\exists x(f)$ evaluate f to be true for *x=1* or *x=0*

Ex:

$$f(x, y, z, w) = xy + z\overline{w} + \overline{w}\,\overline{x}$$

$$\exists x(f) = f_x + f_{\bar{x}} = z\overline{w} + \overline{w} + z\overline{w} + y$$

- Existential quantification of function w.r.t. variable x
- Application: Image computation of sequential circuits