ASIC & FPGA Chip Design:

Synthesis

Mahdi Shabany

Department of Electrical Engineering
Sharif University of technology

Outline

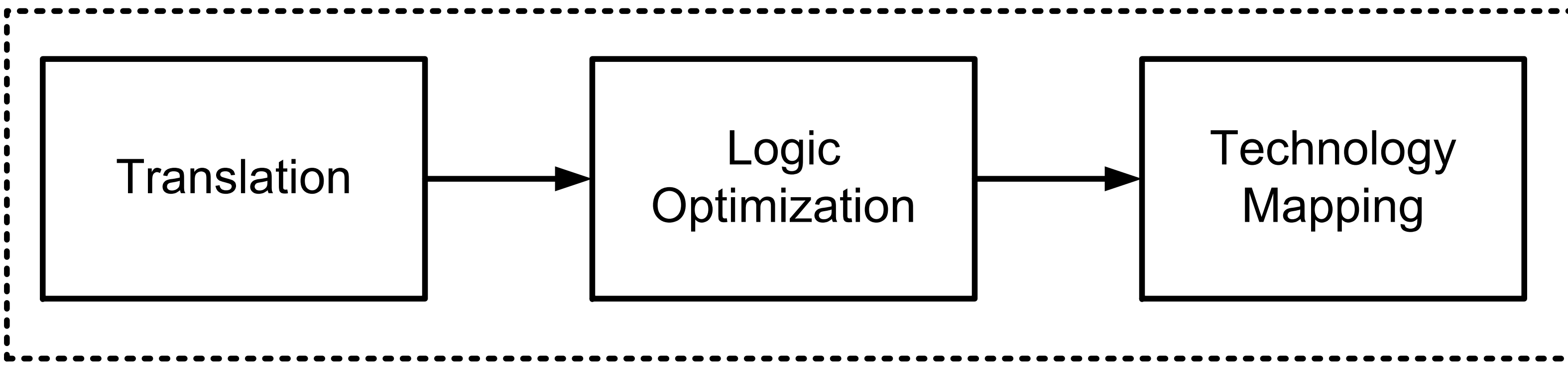
- **Introduction to Synthesis**
- **Digital Logic Basics**
- **Logic Optimization**
  - Two-level logic synthesis
  - Multi-level logic synthesis
- **Technology Mapping**
  - Boolean Satisfiability
  - ASIC/FPGA-oriented Technology Mapping

Outline

- ❑ **Introduction to Synthesis**
- ❑ **Digital Logic Basics**
- ❑ **Logic Optimization**
  - ➢ Two-level logic synthesis
  - ➢ Multi-level logic synthesis
- ❑ **Technology Mapping**
  - ➢ Boolean Satisfiability
  - ➢ ASIC/FPGA-oriented Technology Mapping

Synthesis

- **Synthesis** = Translation + Logic Optimization + Technology Mapping
  - **Translation:** going from RTL to Boolean function
  - **Logic Optimization :** Optimizing and minimizing Boolean function
  - **Technology Mapping (TM):** Map the Boolean function to the target library



Synthesis

Synthesis

- **Synthesis** = Translation + Logic Optimization + Technology Mapping

```
always @ (a, b)
    case ({a,b})
        2'b00: out = 1;
        2'b01: out = 1;
        2'b11: out = 1;
        default: out = 0;
    endcase
```

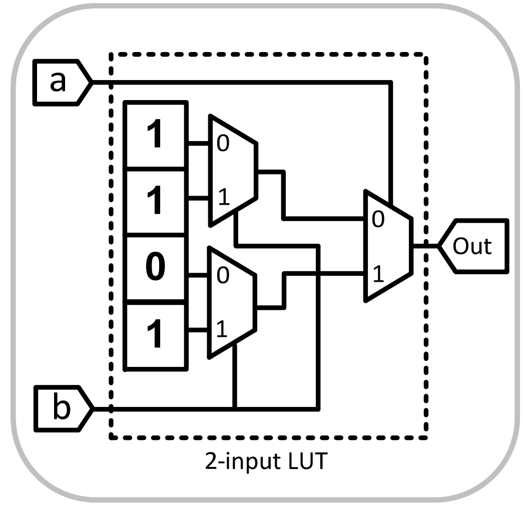
Translation

$$out = \overline{a}b + ab + \overline{a}\,\overline{b}$$

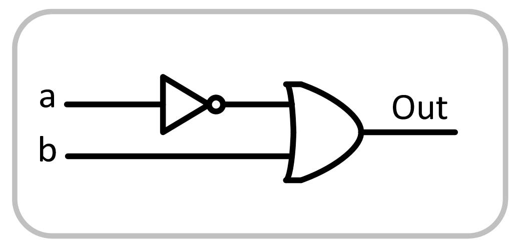
Logic Optimization

$$out = \overline{a} + b$$

Technology Mapping



FPGA



ASIC

Outline

- Introduction to Synthesis
- **Digital Logic Basics**
- Logic Optimization
  - Two-level logic synthesis
  - Multi-level logic synthesis
- Technology Mapping
  - Boolean Satisfiability
  - ASIC/FPGA-oriented Technology Mapping

Digital Logic Basics : Boolean Function

$f(x) : B^n \longrightarrow B$

$B = \{0, 1\}$, $x = (x_1, x_2, \ldots, x_n)$

- $x_1, x_2, \ldots$ are variables
- $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots$ are literals
- each vertex of B^n is mapped to 0 or 1
- the onset of f is a set of input values for which $f(x) = 1$
- the offset of f is a set of input values for which $f(x) = 0$

Digital Logic Basics

- A Boolean function can be represented by:
  - **A truth table**

x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	0

  - **A logic expression**
    - We use logic variables and operators (AND, OR, NOT, XOR, XNOR, NAND, NOR) to express a logical relation b/w input variables and the output function

$$f = x_1 \oplus x_2$$

Digital Logic Basics : SOP

- Sum of Products (SOP):
  - A logic function that is represented as an OR of product (AND) terms :

$$f(x_1, x_2, x_3) = x_1 + \underbrace{x_2x_3}_{\text{cube}} + \underbrace{\bar{x}_1x_2x_3}_{\text{minterm}}$$

Literal

- A <u>literal</u> is a function input (e.g., $\bar{x}_1, x_2$).
- A <u>product term</u> is formed using an AND operation and literals in either true or complemented form.
- A <u>cube</u> is the AND of set of literals
- A <u>minterm</u> is a cube that contains all literals of a logic function
  - A function with K variables has 2^K possible K-literal minterms
- A <u>cover</u> of "**f**" is a set of cubes that represent the logic function "**f**"
  - (e.g., $C = \{x_1, x_2x_3, \bar{x}_1x_2x_3\}$)

Digital Logic Basics : POS

- Product of Sums (POS):
  - A logic function that is represented as an AND of sum (OR) terms :

$$f = (x_1 + x_2)(x_3 + x_2)$$

    - A sum term is formed using an OR operation and literals in either true or complemented form.
    - A maxterm is a sum term that contains all literals of a logic function

- POS can be derived from SOP by DeMorgan's Theorem and double complementation

$$f = x_1 + x_2 x_3$$

$$\overline{f} = \overline{x_1 + x_2 x_3} = \overline{(x_1)} \bullet \overline{(x_2 x_3)} = \overline{x}_1 \bullet (\overline{x}_2 + \overline{x}_3) = \overline{x}_1 \overline{x}_2 + \overline{x}_1 \overline{x}_3$$

$$f = \overline{\overline{f}} = \overline{\overline{x}_1 \overline{x}_2 + \overline{x}_1 \overline{x}_3} = \overline{(\overline{x}_1 \overline{x}_2)} \bullet \overline{(\overline{x}_1 \overline{x}_3)}$$

$$f = (x_1 + x_2)(x_1 + x_3)$$

Digital Logic Basics : Karnaugh Map

❑ Karnaugh Map:

- Variables assigned to rows and columns.
- Adjacent valuations differ by 1 position.
- Form product terms by creating groups with 2^k 1s that are adjacent to one another

cd \ ab	00	01	11	10
00	0	0	0	1
01	0	1	1	1
11	0	1	1	0
10	1	1	0	1

$$f = bd + a\overline{b}\,\overline{c} + \overline{a}c\overline{d} + \overline{b}c\overline{d}$$

Digital Logic Basics : Implementation Cost

❑ Cost of an Implementation:

$$\boxed{\text{Cost of implementation = \# of inputs + \# of gates (except NOTs)}}$$

❖ **Example:**

$$f = \underbrace{\underbrace{bd}_{3} + \underbrace{a\bar{b}\bar{c}}_{4} + \underbrace{\bar{a}c\bar{d}}_{4} + \underbrace{\bar{b}c\bar{d}}_{4}}_{5}$$

- Cost = 3 + 4 + 4 + 4 + 5 = 20

Digital Logic Basics : Canonical Form

- **Canonical Form:**
  - A form of Boolean logic function representation is said to be canonical if and only if for each logic function there exists a unique representation in the given form
  - Function is described using its equivalent minterms

Example:

- **Canonical** $f(a,b,c) = a\overline{b}c + abc$
- **Non-canonical** $f(a,b,c) = a\overline{b}c + abc + bc$
  - Because it has a cube that does not include all the literals

- Any Boolean logic can be represented in a canonical form

Outline

- Introduction to Synthesis
- Digital Logic Basics
- **Logic Optimization**
  - Two-level logic synthesis
  - Multi-level logic synthesis
- Technology Mapping
  - Boolean Satisfiability
  - ASIC/FPGA-oriented Technology Mapping

Two-Level Logic Synthesis : Cubical Notation

Cubical Notation:

- A different way to represent a product term
  - "0" : to represent inverted variable
  - "1" : to represent a variable in true form
  - "X" : to represent a variable not used in the product term

abcd		p-term
X1X1	→	
100X	→	
0X10	→	
X010	→	

f = {X1X1, 100X, 0X10, X010}

↓

(Easier to implement in a computer program)

Two-Level Logic Synthesis : Cubical Notation

Cubical Notation:

- A different way to represent a product term
  - "0" : to represent inverted variable
  - "1" : to represent a variable in true form
  - "X" : to represent a variable not used in the product term

abcd		p-term
X1X1	$\longrightarrow$	bd
100X	$\longrightarrow$	$a\bar{b}\bar{c}$
0X10	$\longrightarrow$	$\bar{a}c\bar{d}$
X010	$\longrightarrow$	$\bar{b}c\bar{d}$

f = {X1X1, 100X, 0X10, X010}

$\Downarrow$

(Easier to implement in a computer program)

Two-Level Logic Synthesis: Quine-McCluskey Method

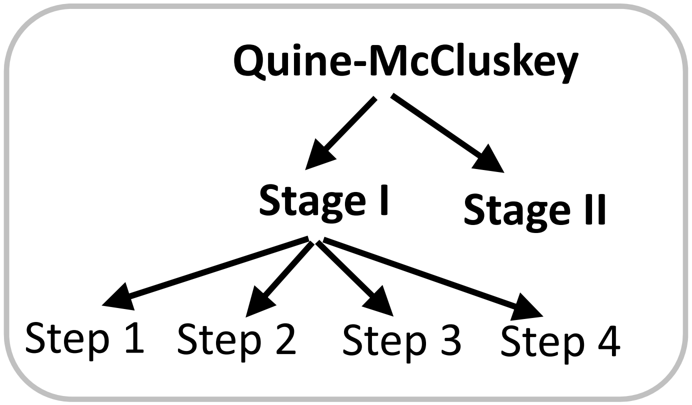
❑ How do we use cubical notation to synthesize logic functions?

- Quine-McCluskey method
  - Willard Quine (1908-2000)) and Edward McCluskey (1928-present)

❑ Quine-McCluskey method: (2 Stages)

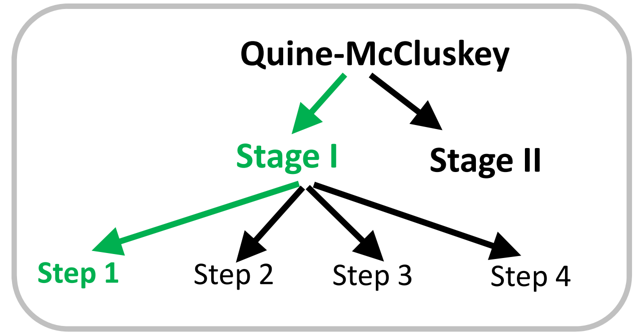
Stage I : Take minterms for a function (Canonical) and form Prime Implicants (PIs).
Stage II : Pick minimum Prime Implicants to generate a cover.

A cover of a function is a set of cubes that represent that logic function



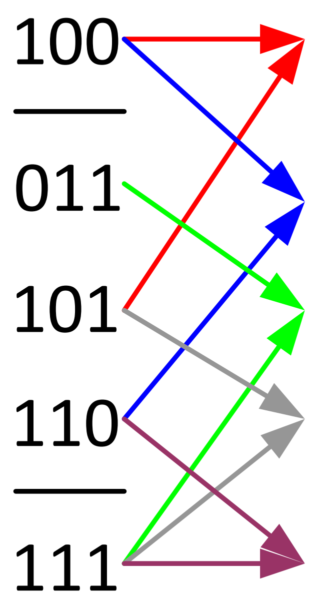
Quine-McCluskey Method : Stage I – Step 1

- **Stage I** of Quine-McCluskey method has four steps:



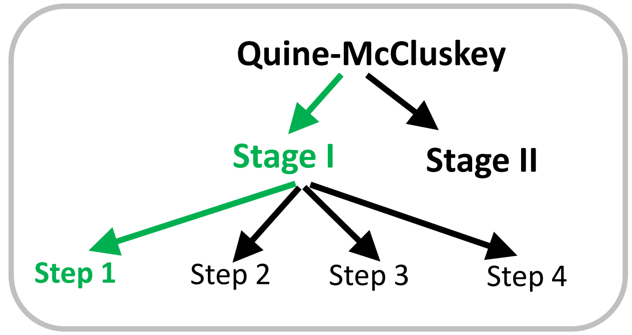
$$f(a,b,c)=\sum m(3,4,5,6,7)$$

Step 1: List all minterms grouped in increasing order of the number of 1s they contain. Take cubes from adjacent groups to form larger product terms.



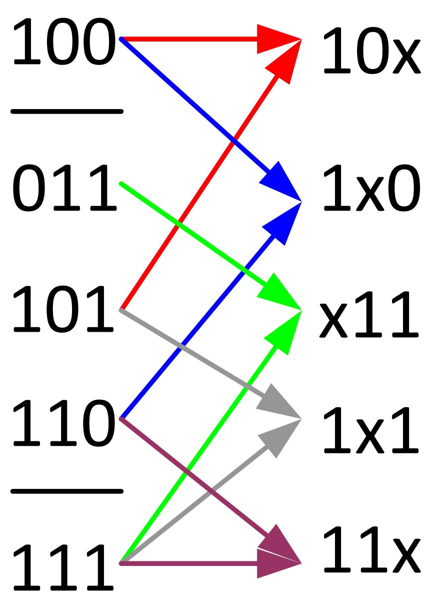
Quine-McCluskey Method : Stage I – Step 1

- **Stage I** of Quine-McCluskey method has four steps:



$$f(a,b,c)=\sum m(3,4,5,6,7)$$

Step 1: List all minterms grouped in increasing order of the number of 1s they contain. Take cubes from adjacent groups to form larger product terms.



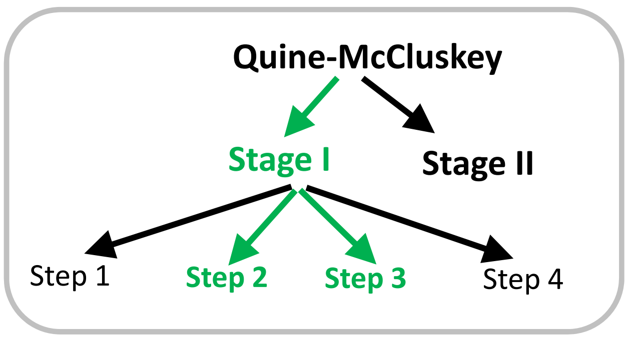
Quine-McCluskey Method : Stage I – Step 2/3

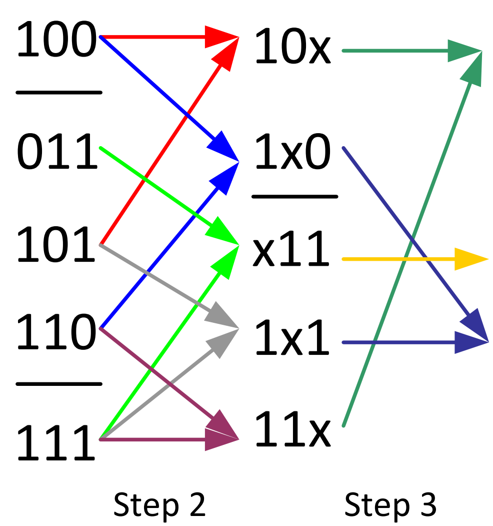
Step 2: Take original set of cubes and the newly created ones.
Remove any cube from the set that is completely covered by any other cube.

- In this case we remove all minterms and use only the newly created cubes.

Step 3: Group cubes in the increasing order of **non-zero positions.** By taking terms from adjacent groups form larger product terms.

- **Rules:** X's must match and the other position must differ by exactly one position (one cube has a 1 where the other has a 0).





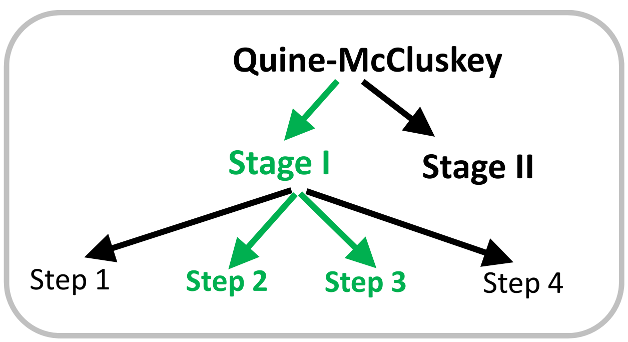
Quine-McCluskey Method : Stage I – Step 2/3

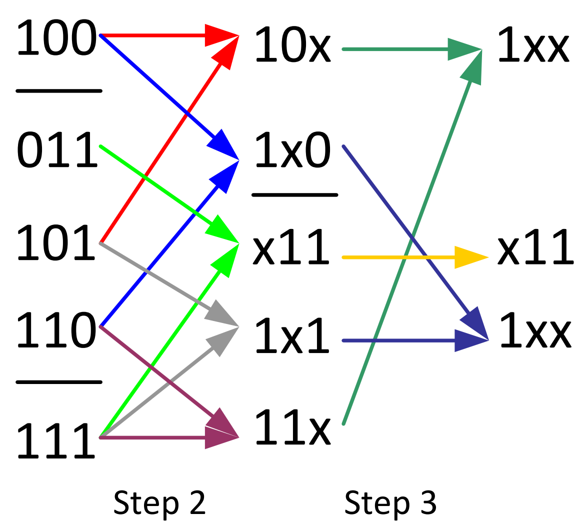
Step 2: Take original set of cubes and the newly created ones.
Remove any cube from the set that is completely covered by any other cube.

- In this case we remove all minterms and use only the newly created cubes.

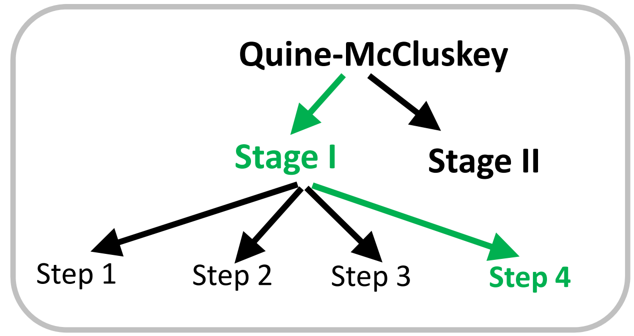
Step 3: Group cubes in the increasing order of **non-zero positions.** By taking terms from adjacent groups form larger product terms.

- **Rules:** X's must match and the other position must differ by exactly one position (one cube has a 1 where the other has a 0).



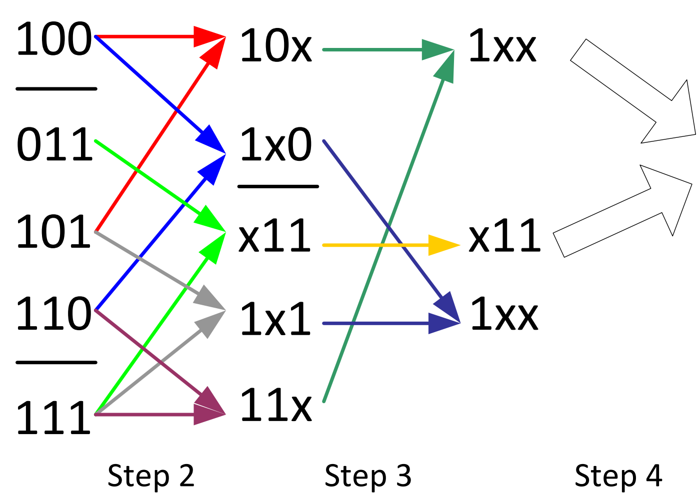


Quine-McCluskey Method : Stage I – Step 4

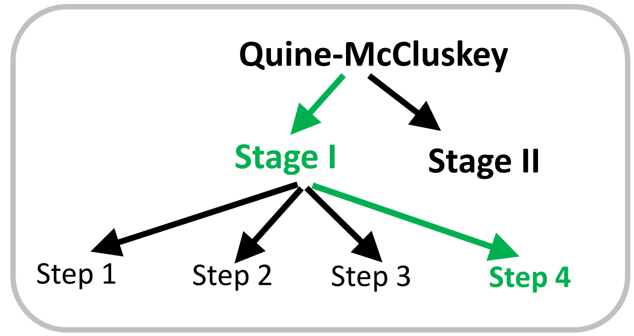


Step 4: Repeat steps 2-4 until all PIs are generated.

- In our case we are done, because 1xx and x11 are the PIs for this function.
- PI set = { 1xx, x11 }

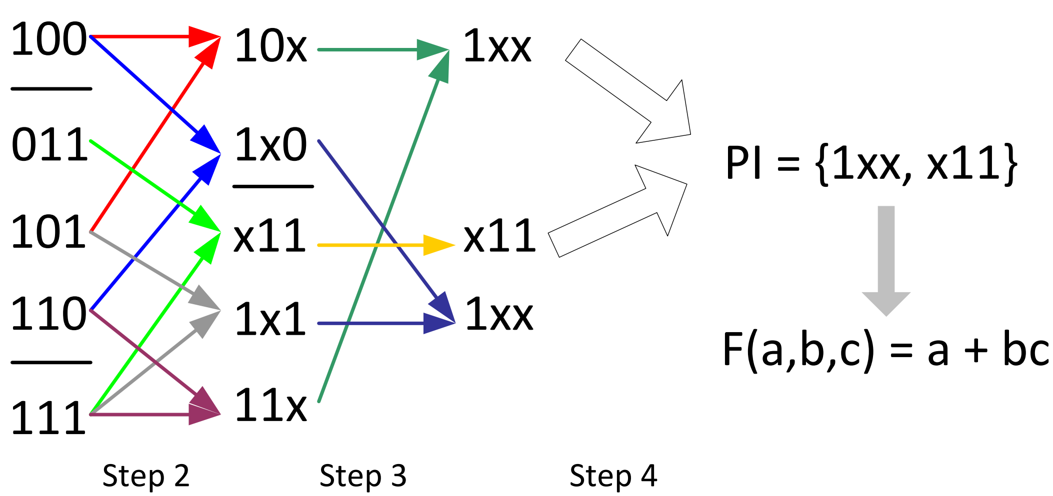


Quine-McCluskey Method : Stage I – Step 4

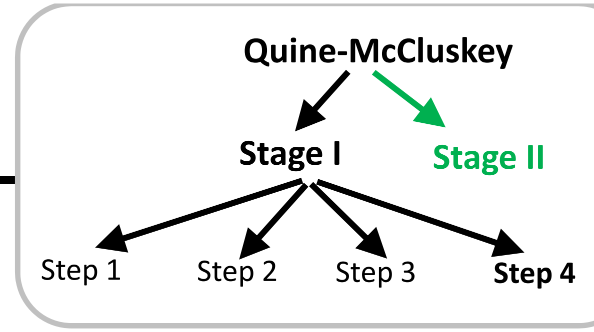


Step 4: Repeat steps 2-4 until all PIs are generated.

- In our case we are done, because 1xx and x11 are the PIs for this function.
- PI set = { 1xx, x11 }



Quine-McCluskey Method : Stage II



- **Stage II:** Build a table list the minterms that are covered by each PI
  - (Prime Implicant Cover Table)
- Pick the essential prime implicants (columns with only one check mark)
  - 1XX (the only one to cover 4, 5, 6, 7), X11 (the only one to cover 3)

Cube	3	4	5	6	7
1xx		X	X	X	X
x11	X				

PI = {1xx, x11}

Done!

Quine-McCluskey Method : Stage I – (Step 1-4)

❖ **Example:** $f(a,b,c,d,e)=\sum m(0,2,3,4,6,7,9,12,13,15,16,23,24,25,29,31)$

00000

00010
00100
10000

00011
00110
01001
01100
11000

00111
01101
11001

01111
10111
11101

11111

000x0
00x00
x0000

0001x
00x10
001x0
0x100
1x000

00x11
0011x
01x01
x1001
0110x
1100x

0x111
011x1
x0111
x1101
11x01

x1111
1x111
111x1

Final PIs

00xx0
x0000

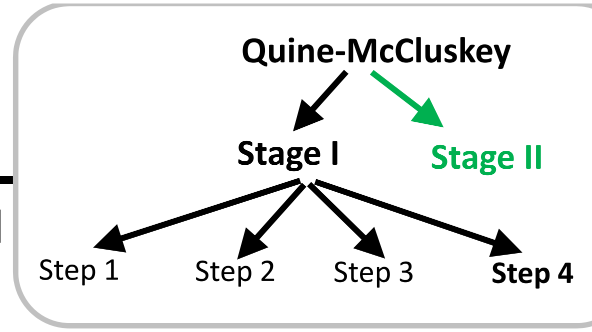
00x1x
0x100
1x000

x1x01
0110x
1100x

xx111
x11x1

When starting with minterms, we need to only consider the combination of cubes that create a larger cube (more x's).

Quine-McCluskey Method : Stage II

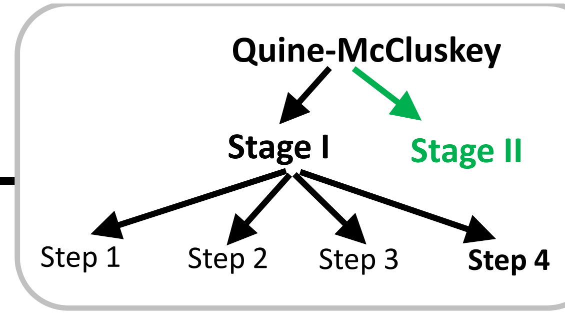


- **Stage II:** Build a table list the minterms that are covered by each PI
  - (Prime Implicant Cover Table)
- Pick the essential prime implicants (columns with only one check mark)
  - 00x1x, x1x01, xx111 (the only one to cover 23)

Cube	0	2	3	4	6	7	9	12	13	15	16	23	24	25	29	31
00xx0	X	X		X	X											
x0000	X										X					
00x1x		X	X		X	X										
0x100				X				X								
1x000											X		X			
x1x01							X		X					X	X	
0110x								X	X							
1100x													X	X		
xx111						X				X		X				X
x11x1									X	X					X	X

PI = {00x1x, x1x01, xx111}

Quine-McCluskey Method : Stage II



- ❑ Remove the columns covered by the essential prime implicants
  - ➢ Columns: 2, 3, 6, 7 covered by 00x1x
  - ➢ Columns: 9, 13, 25, 29 by x1x01
  - ➢ Columns: 15, 23, 31 by xx111

Remove used PIs and covered minterms from the table

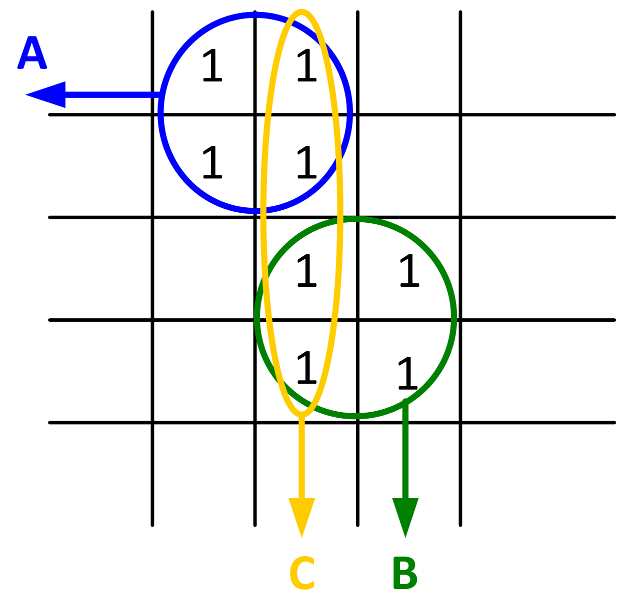
Cube	0	2	3	4	6	7	9	12	13	15	16	23	24	25	29	31
00xx0	X	X		X	X											
x0000	X										X					
00x1x		X	X		X	X										
0x100				X				X								
1x000											X		X			
x1x01							X		X					X	X	
0110x								X	X							
1100x													X	X		
xx111						X				X		X				X
x11x1									X	X					X	X

PI = {00x1x, x1x01, xx111}

Quine-McCluskey Method : Stage II

- Notice that cube x11x1 does not cover any of the remaining minterms. So remove it.

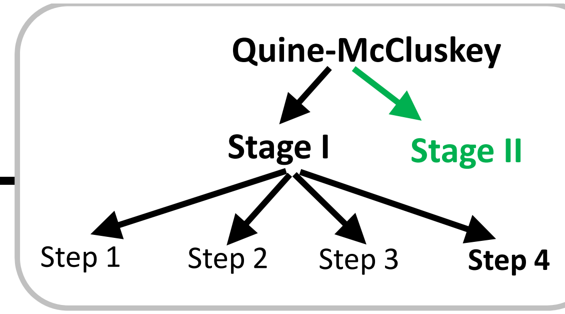
	0	4	12	16	24
00xx0	X	X			
X0000	X			X	
0x100		X	X		
1x000				X	X
0110x			X		
1100x					X
x11x1					



If we select A and B, then C does not do anything!

PI = {00x1x, x1x01, xx111}

Quine-McCluskey Method : Stage II



❑ Definition 1: Row dominance

- Row A in the covering table dominates another row B if and only if row A covers at least the same set of columns as row B, and the PI in row B has equal or greater cost than the PI in row A.
- Notice that 1x000 dominates 1100x, 0x100 dominates 0110x., so:

	0	4	12	16	24
00xx0	X	X			
X0000	X			X	
0x100		X	X		
1x000				X	X
0110x			X		
1100x					X

	0	4	12	16	24
00xx0	X	X			
X0000	X			X	
0x100		X	X		
1x000				X	X

PI = {00x1x, x1x01, xx111}

Quine-McCluskey Method : Stage II

Definition 1: Column dominance

- Column i dominates column j if and only if column i is covered by at least the same set of cubes as column j. In such cases we remove the column i from the table.
- Notice that column 4 dominates column 12, and column 16 dominates 24, so:

	0	4	12	16	24
00xx0	X	X			
X0000	X			X	
0x100		X	X		
1x000				X	X

	0	12	24
00xx0	X		
X0000	X		
0x100		X	
1x000			X

PI = {00x1x, x1x01, xx111, 0x100, 1x00}

Essential PIs

Quine-McCluskey Method : Stage II

- By row dominance, x0000 is removed, so:
  - It covers the same number of columns as 00xx0, but it has a larger cost

	0	12	24
00xx0	X		
X0000	X		

	0	12	24
00xx0	X		

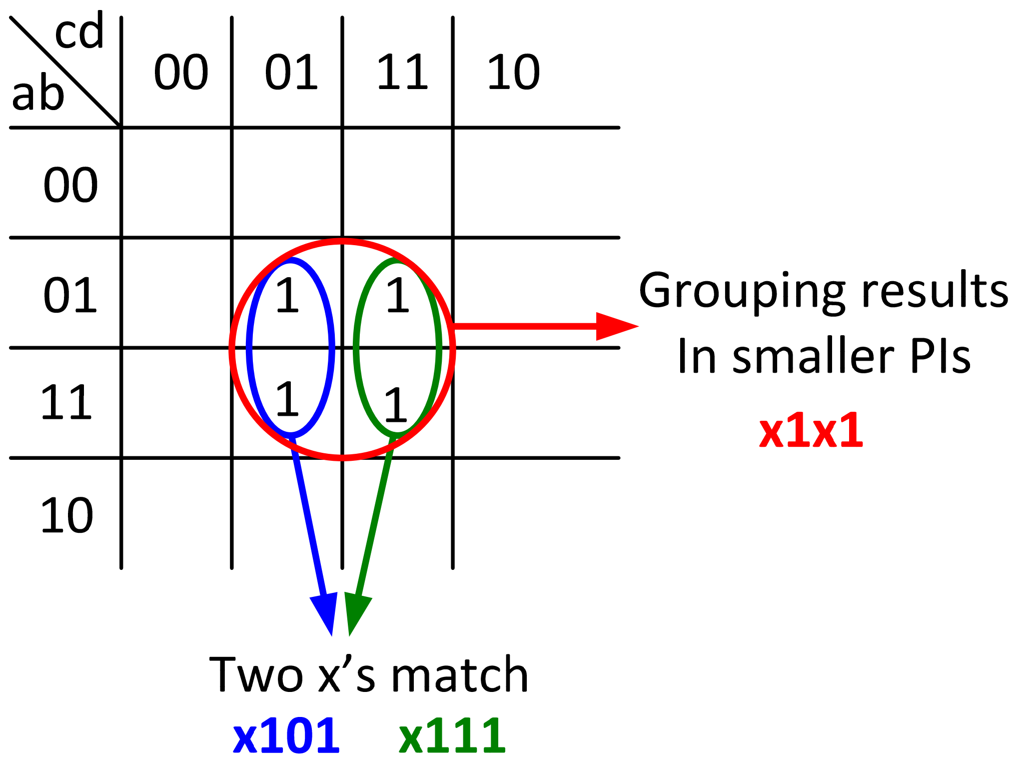
PI = {00x1x, x1x01, xx111, 0x100, 1x00, 00xx0}

Quine-McCluskey Method : Summary

1. Generate all PIs starting with minterms
2. Create a prime implicant covering table. List PIs in the rows and function minterms in the columns.
3. For each PI indicate which minterms it covers by putting a checkmark in the corresponding column.
4. For each column covered by exactly one PI add the corresponding PI to your cover, removing the PI from the table, as well as any columns it covers.
5. Apply concepts of row and column dominance to reduce the table.
6. Repeat steps 4-6 until the table cannot be further reduced.

Quine-McCluskey Method : Note 1

- Why should x’s match?
  - Matching x’s is equivalent to grouping PIs in the Karnaugh map.



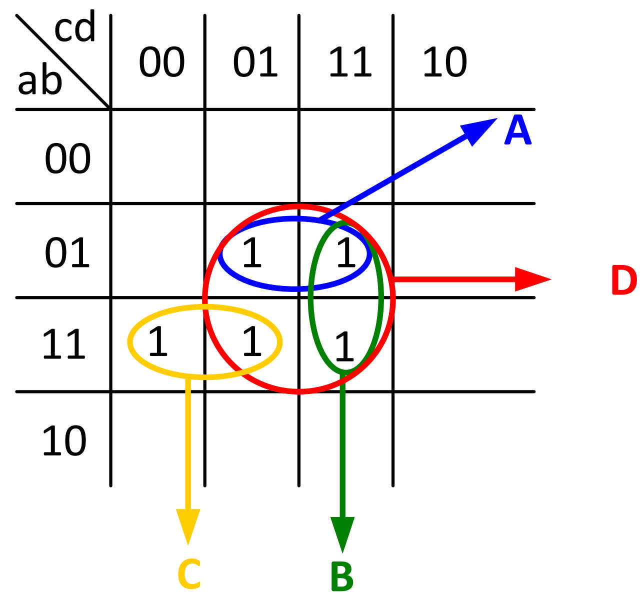
Quine-McCluskey Method : Note 2

- When starting with minterms (Canonical form), the above procedure works fine.
- Otherwise, it is necessary to represent the function as a sum of minterms first
- Example:

$$f = \underbrace{\bar{a}bd}_{A} + \underbrace{bcd}_{B} + \underbrace{ab\bar{c}}_{C} \Rightarrow \text{PI set} = \{\,01x1,\ x111,\ 110x\}$$

  - Not possible to combine/reduce the above PI set (no x's match)
  - Represent "f" as a sum of minterms (canonical form)

$$f = ab\bar{c}\bar{d} + ab\bar{c}d + abcd + \bar{a}b\bar{c}d + \bar{a}bcd$$



Outline

- ❑ Introduction to Synthesis
- ❑ Digital Logic Basics
- ❑ **Logic Optimization**
  - ➢ Two-level logic synthesis
  - ➢ Multi-level logic synthesis
- ❑ Technology Mapping
  - ➢ Boolean Satisfiability
  - ➢ ASIC/FPGA-oriented Technology Mapping

Multi-Level Logic Synthesis

- Two-level logic synthesis is effective and mature
- Two-level logic synthesis is directly applicable to PLAs and PLDs

However, …

- There are many functions that are too expensive to implement in two-level forms (too many product terms!)
- Two-level implementation constrains layout (AND-plane, OR-plane)

Multi-level logic synthesis may be employed

- Rule of thumb:
  - Two-level logic is good for control logic
  - Multi-level logic is good for data path or random logic

Multi-Level Logic Synthesis

- Multi-level logic synthesis:
  - Decompose a logic function into smaller functions
  - A simple tool to do this is called Shannon's Decomposition
    - Claude Shannon 1916-2001
- Shannon's Expansion Theorem:

Any logic function $f(x_1, x_2, \ldots, x_n)$ can be expanded in the form of:

$$x_k . f(x_1, x_2, x_{k-1}, 1, x_{k+1}, \ldots, x_n) + \overline{x}_k . f(x_1, x_2, x_{k-1}, 0, x_{k+1}, \ldots, x_n)$$

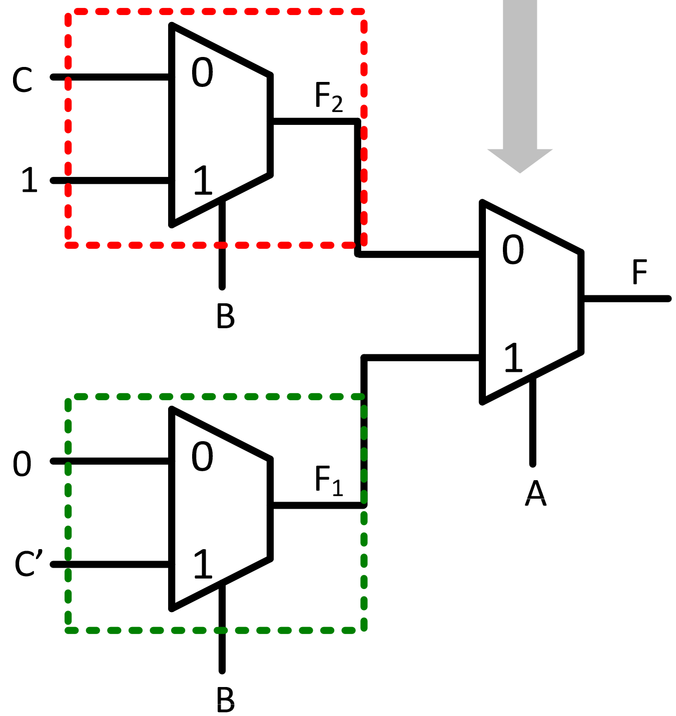
$$= x_k . f_{x_k} + \overline{x}_k . f_{\overline{x}_k}$$

- $f_{x_k} = f(x_k = 1, \ldots)$: 1- cofactor
- $f_{\overline{x}_k} = f(x_k = 0, \ldots)$: 0- cofactor

Multi-Level Logic Synthesis

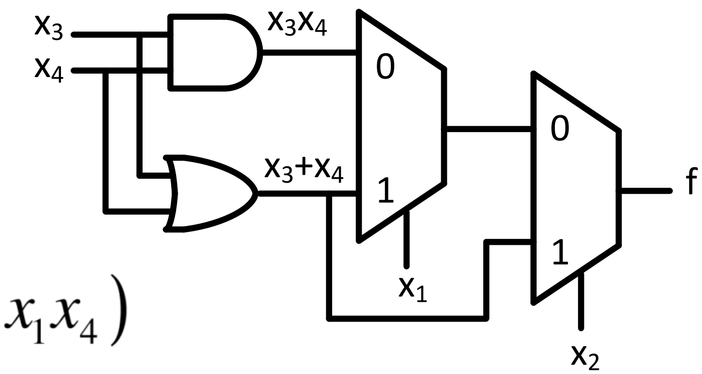
- Example:
  - $F(A, B, C) = A'B + ABC' + A'B'C$

    $= A(BC') + A'(B+B'C) = A.F_1 + A'.F_2$
  - $F_2 = B+B'C = B.1 + B'.C$
  - $F_1 = BC' = BC' + B'.0$



Multi-Level Logic Synthesis

❖ Example:



$$f = \overline{x}_1\overline{x}_2x_3x_4 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4$$

$$f = x_2\left(x_1x_3 + x_1x_4 + x_3 + x_4\right) + \overline{x}_2\left(\overline{x}_1x_3x_4 + x_1x_3 + x_1x_4\right)$$

$$f = x_2\left(x_3 + x_4\right) + \overline{x}_2\left(\overline{x}_1x_3x_4 + x_1\left(x_3 + x_4\right)\right)$$

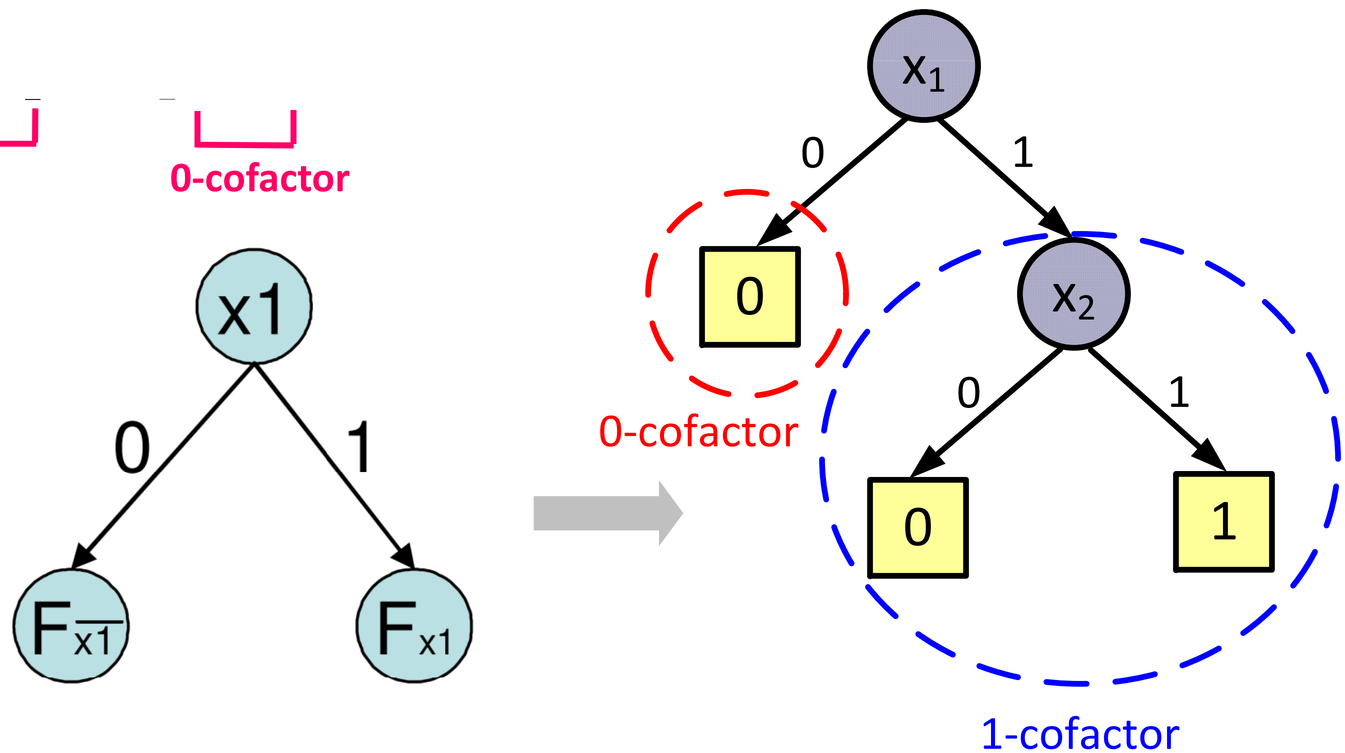
- We found out that logic expression (x_3+x_4) appears in multiple places. Hence, this knowledge could be used to simplify the implementation of this circuit.
- Keeping track of this type of relationships in a logic function can be tedious in a large expression.
- Can we represent this information in a better way?
  - Yes, we can use **Binary Decision Diagrams (BDDs)**

Binary Decision Decomposition (BDD)

- Let us decompose a function "$f=x_1x_2$" with respect to x_1:

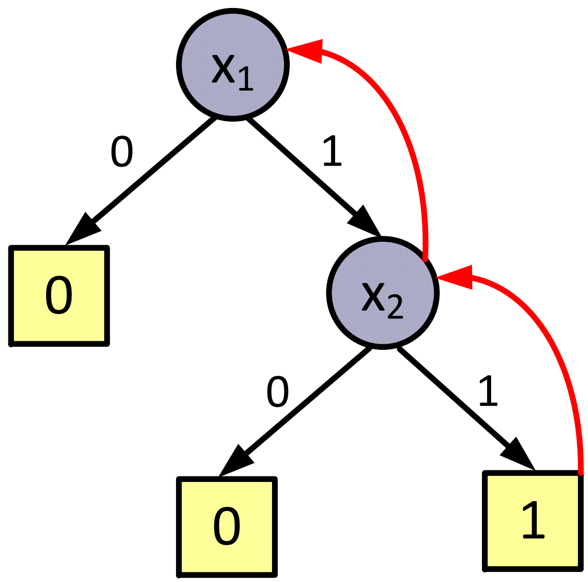
$$f = x_1 . f_{x_1} + \overline{x}_1 . f_{\overline{x}_1}$$

1-cofactor 0-cofactor

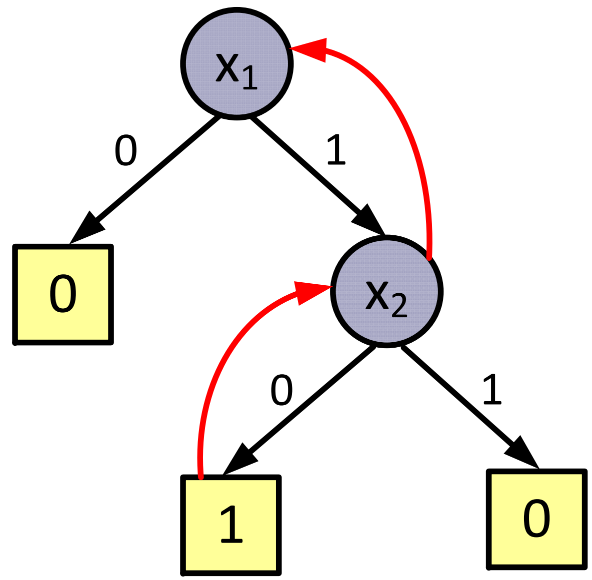


Binary Decision Decomposition (BDD)

- To derive the function from BDD, start from the bottom to top
- Start only with terminal nodes 1
- If see edge “1” use the variable otherwise its complement



$f = x_1x_2$



$f = x_1\overline{x}_2$

Binary Decision Decomposition (BDD)

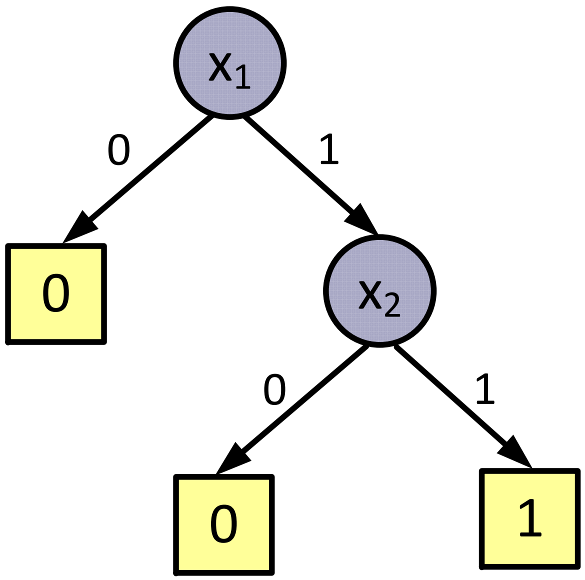
- A binary decision diagram represents a logic function by using Shannon's decomposition to decompose a function into cofactors, one variable at a time:

$$f = x_k . f_{x_k} + \overline{x}_k . f_{\overline{x}_k}$$

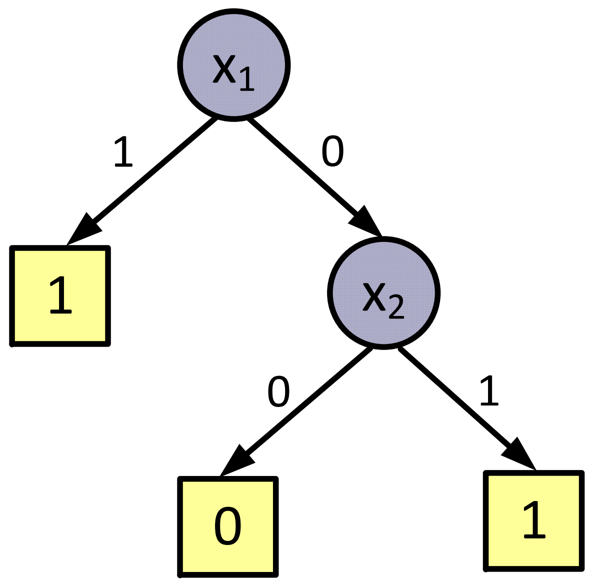
- The decomposition steps are then represented as a directed graph G = (V,E), where:
  - **V** is a set of vertices. Each vertex is associated with a variable or a constant 0/1.
  - **E** is a set of directed edges. Each edge is assigned a label of 0 or 1. A 0 edge always points to a 0-cofactor and a 1 edge points to a 1-cofactor.

Binary Decision Decomposition (BDD)

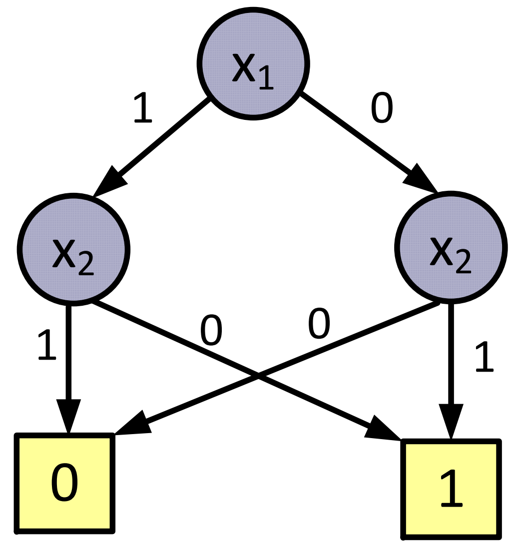
- BDD of some basic logic functions:



$f = x_1x_2$



$f = x_1+x_2$

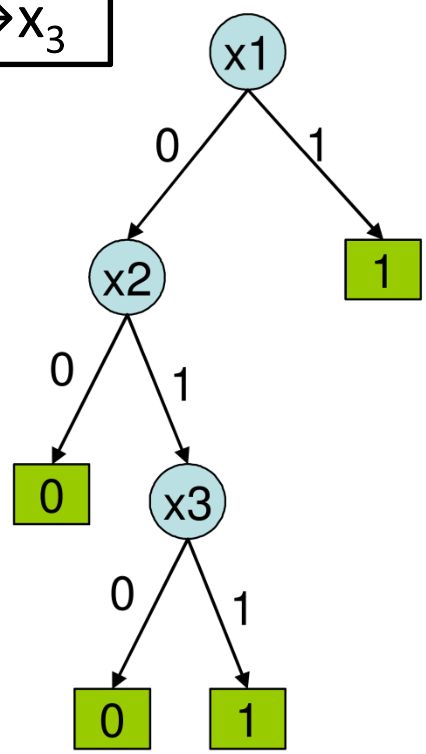


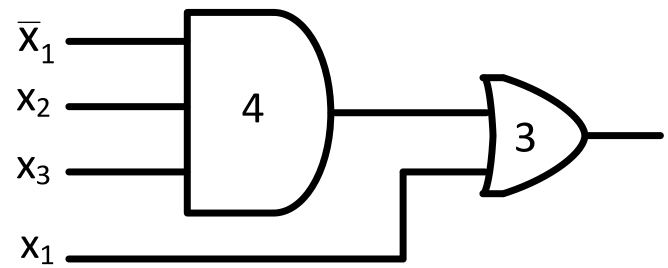
$f = x_1 \oplus x_2$

BDD: Decomposition Order Matters

- Example: $f = x_1 + x_2x_3 = x_1(1) + \bar{x}_1x_2x_3$

$x_1 \rightarrow x_2 \rightarrow x_3$



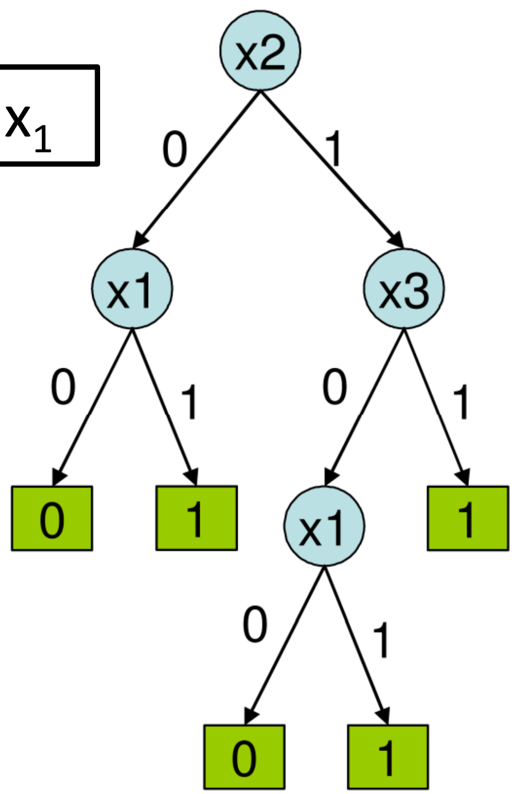


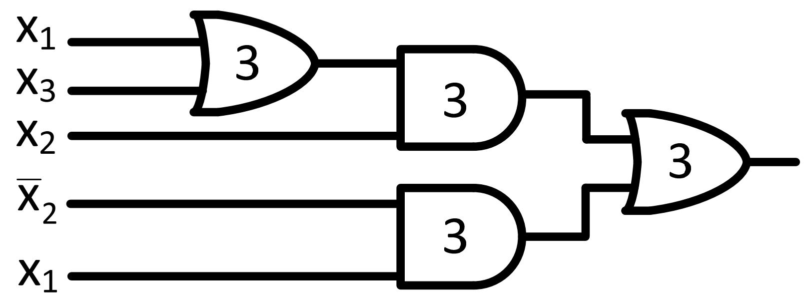
Cost : 4 + 3 = 7

BDD: Decomposition Order Matters

- Example: $f = x_1 + x_2x_3$

$x_2 \rightarrow x_3 \rightarrow x_1$





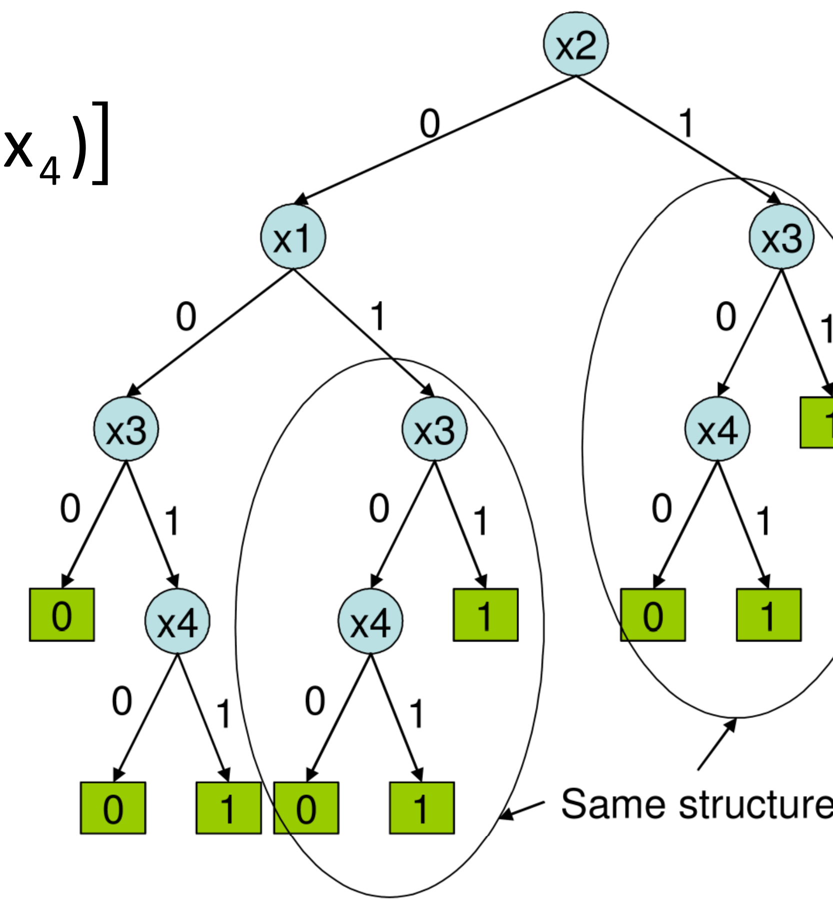
Cost : 3 + 3 + 3 + 3 = 12

Different order of decomposition => different size/cost of the diagram

Ordered Binary Decision Decomposition (OBDD)

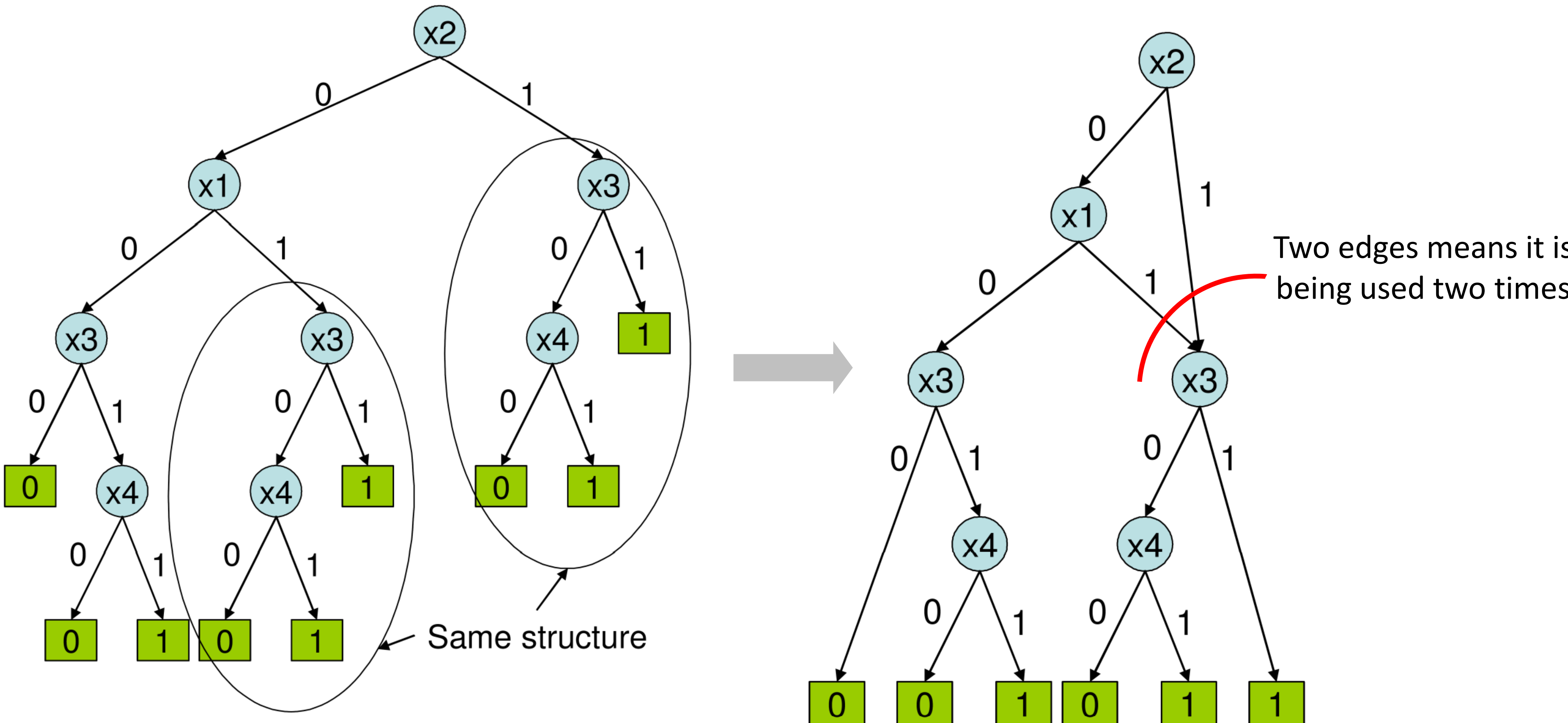
- We could impose a decomposition order by fixing the sequence of variable with respect to which we decompose a function **(Ordered BDD = OBDD)**
- Let us consider the following function:

$$f = x_2(x_3 + x_4) + \bar{x}_2\left[\bar{x}_1(x_3 x_4) + x_1(x_3 + x_4)\right]$$



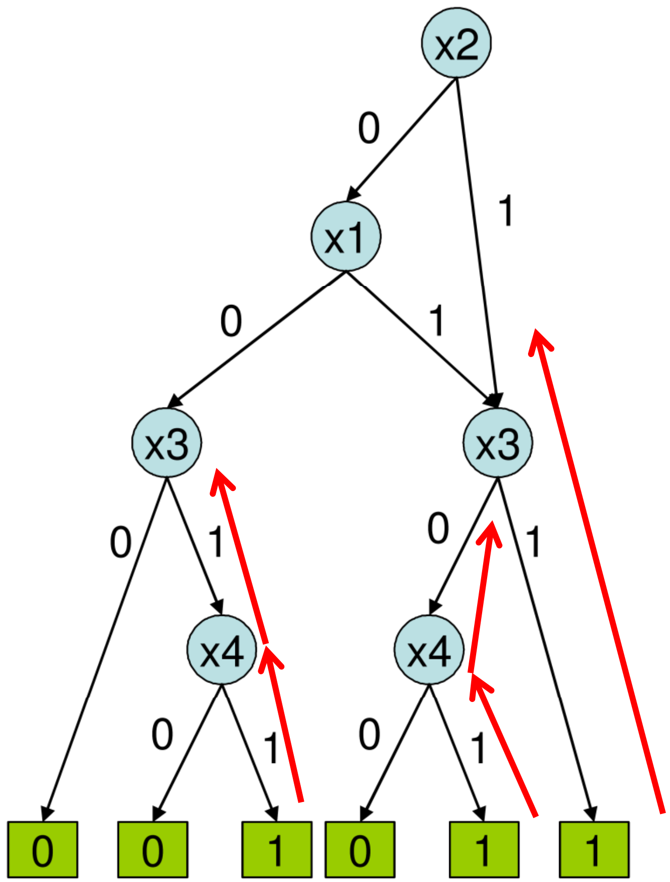
Reducing OBDDs

- Similar structure can be used to reduce the OBDD



Reducing OBDDs

- Cost of this OBDD:

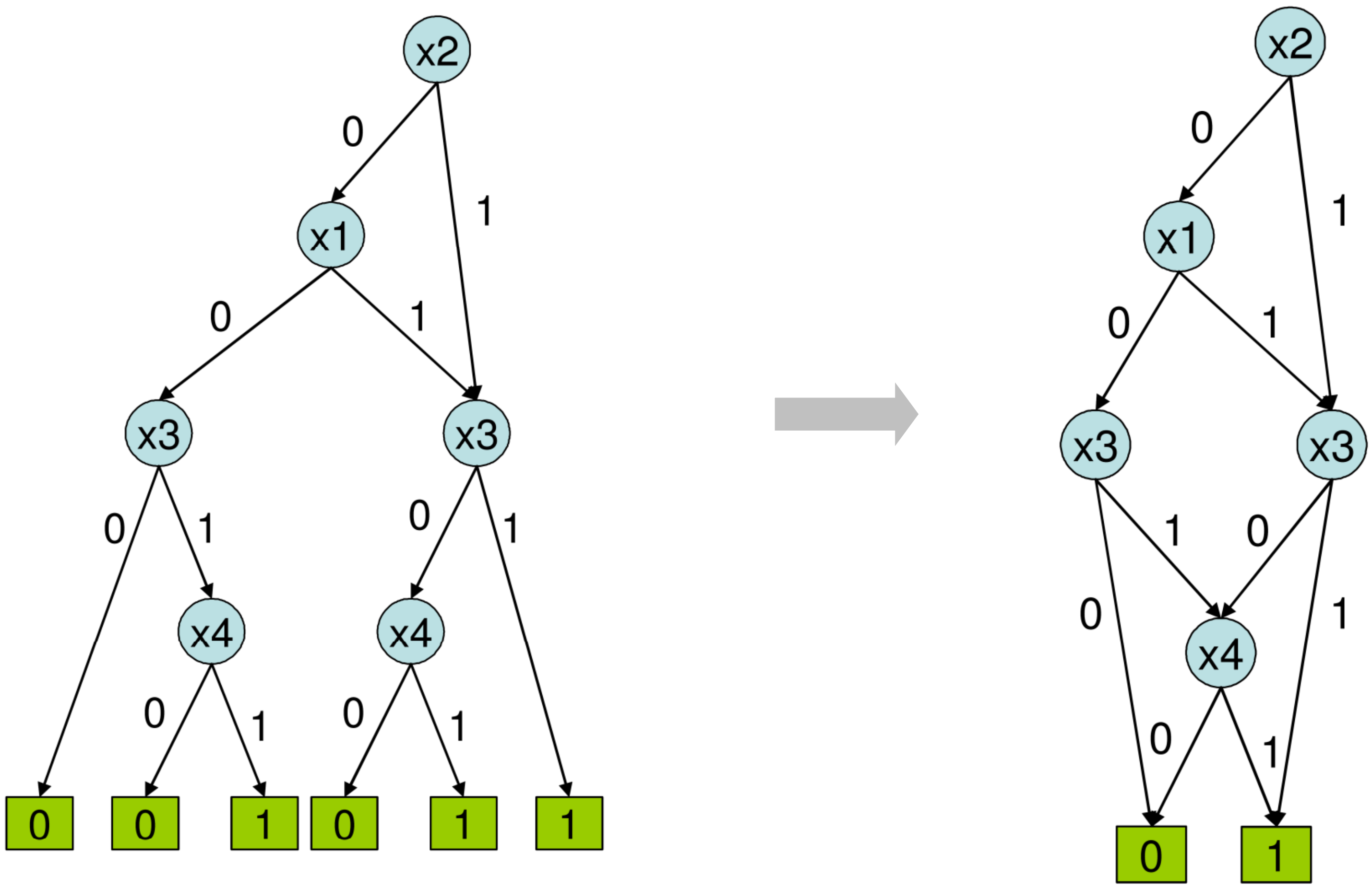


$$f = x_4 x_3 \bar{x}_1 \bar{x}_2 + (x_4 + x_3)(x_1 + x_2)$$

Cost : 5 + 3 + 3 + 3 + 3 = 17

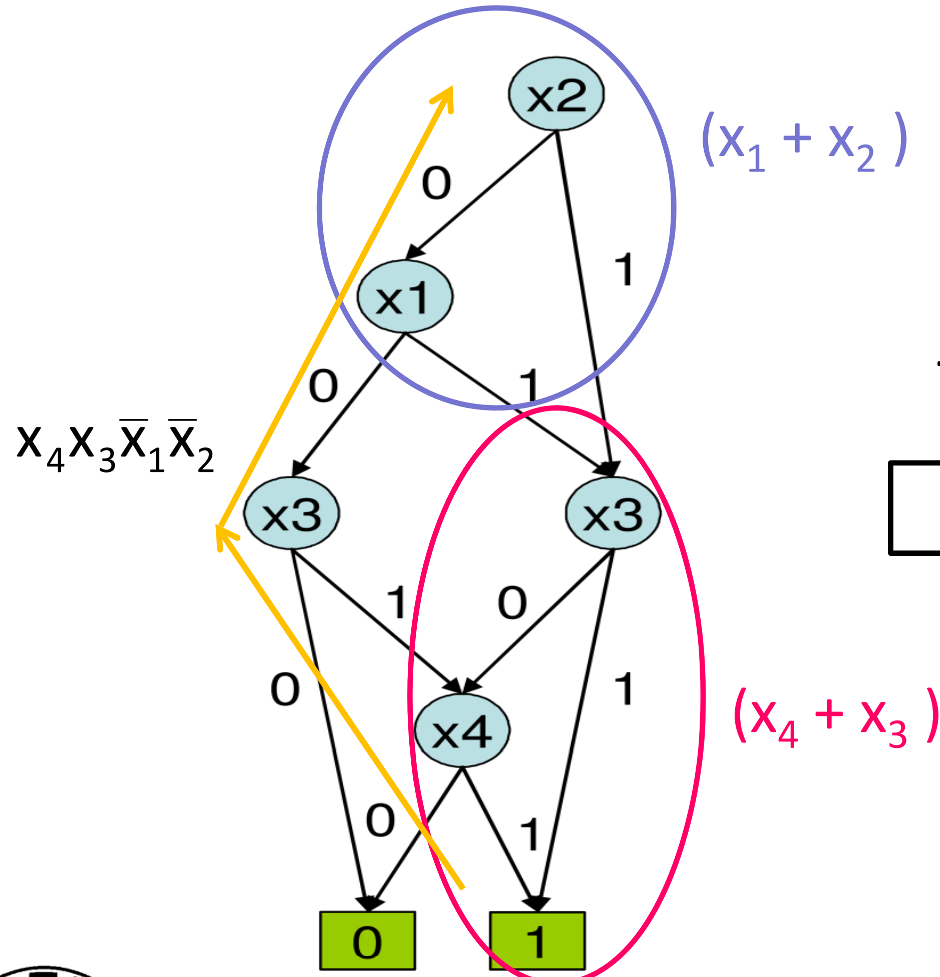
Reduced Ordered BDD (ROBDD)

❑ Bottom-up merging of the isomorphic graphs together to simplify the OBDD into a **Reduced Ordered BDD** (ROBDD)



Reduced Ordered BDD (ROBDD)

- Thus, bottom-up merging the graphs together will reduce the cost

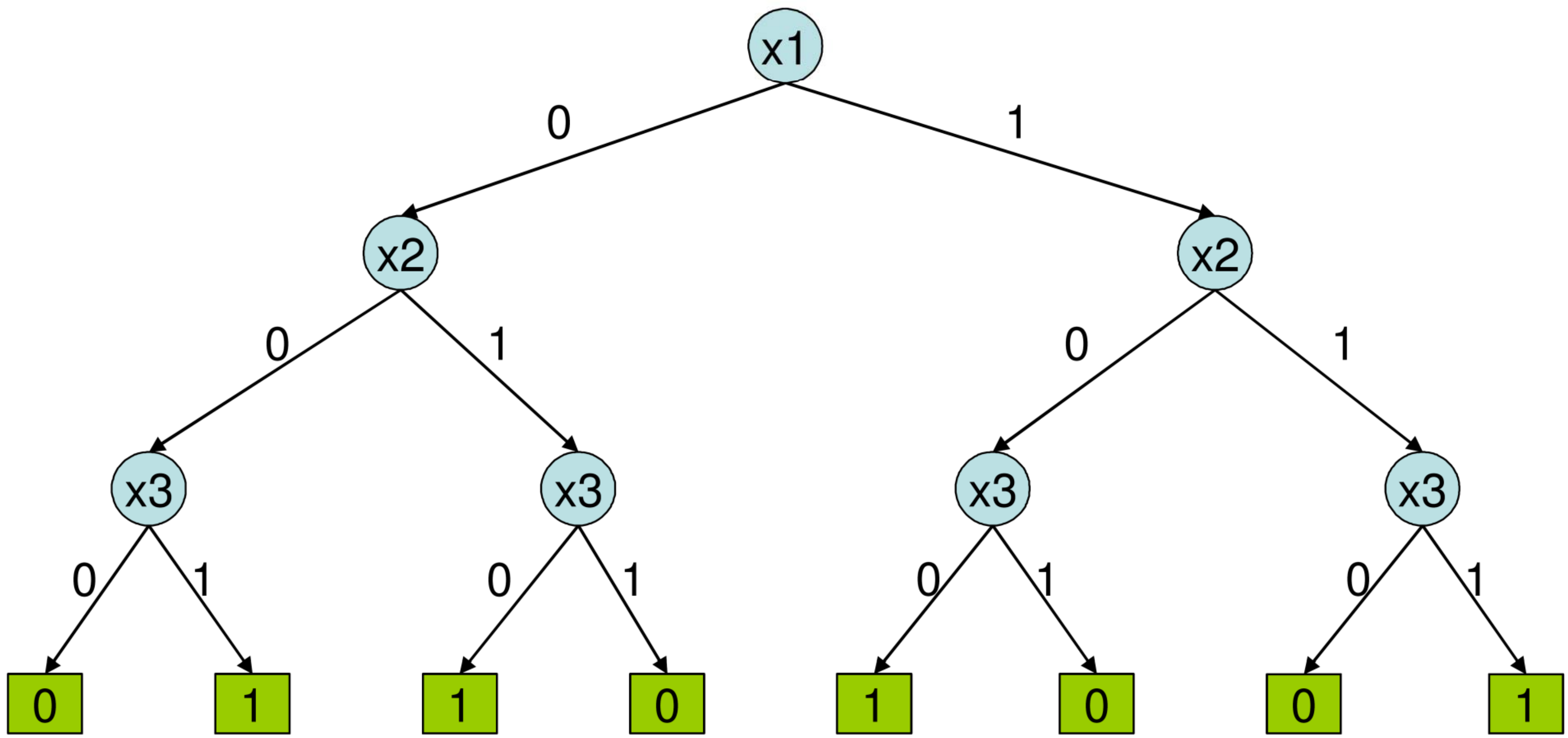


$$f = x_4 x_3 \bar{x}_1 \bar{x}_2 + (x_3 + x_4)(x_1 + x_2)$$

$\boxed{\text{Cost : } 5 + 3 + 3 + 3 + 3 = 17}$

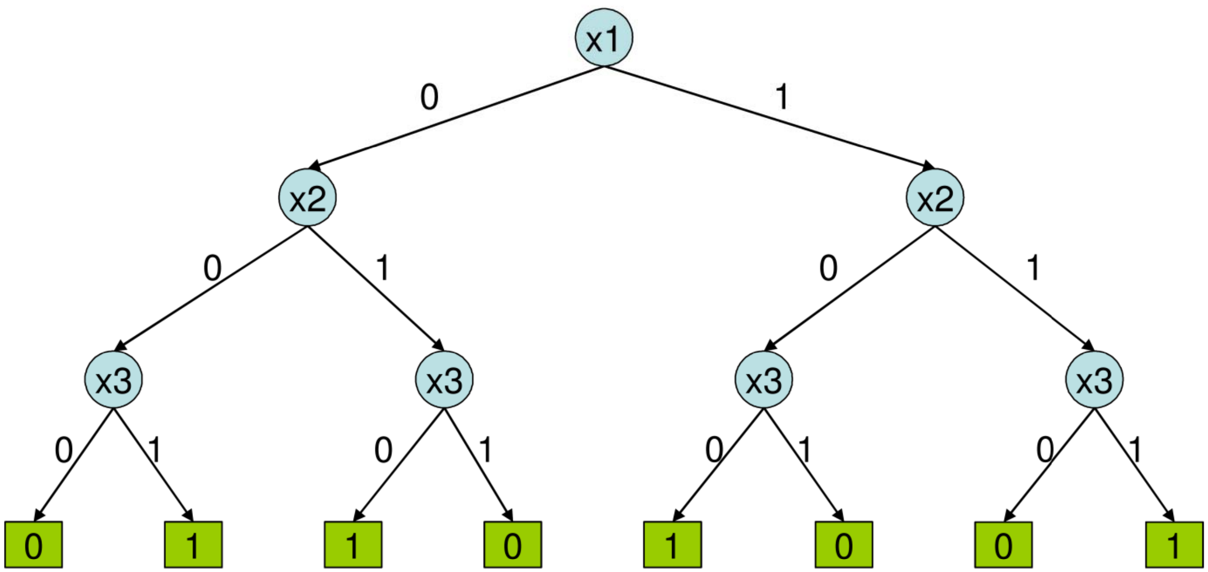
Reduced Ordered BDD (ROBDD)

- **Example:** $f = x_1 \oplus x_2 \oplus x_3$

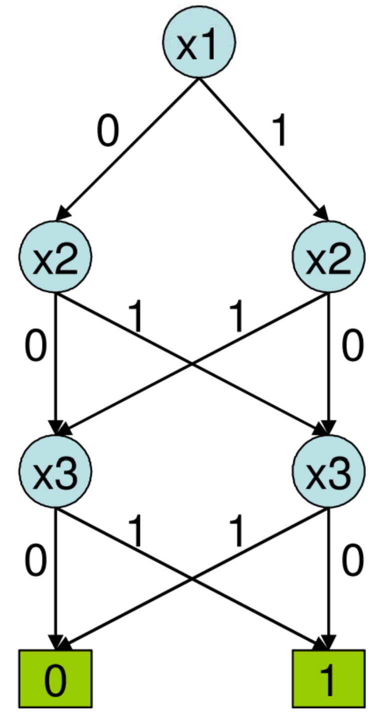


Reduced Ordered BDD (ROBDD)

❖ **Example:** $f = x_1 \oplus x_2 \oplus x_3$



15 nodes



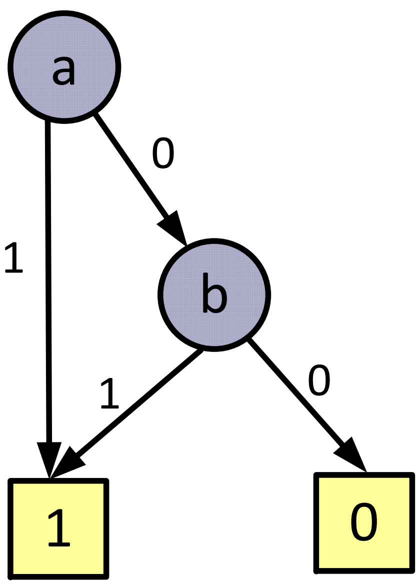
7 nodes

BDD of f <op> g

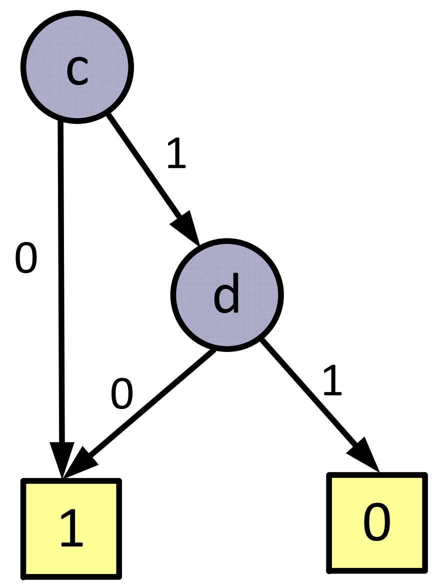
- Goal: take BDDs for functions "f" and "g" and produce a BDD for **f <op> g.**
- **Case 1:** functions f and g have distinct support (i.e., $sup(f) \cap sup(g) = \phi$)

Example:

$$f = a + b$$

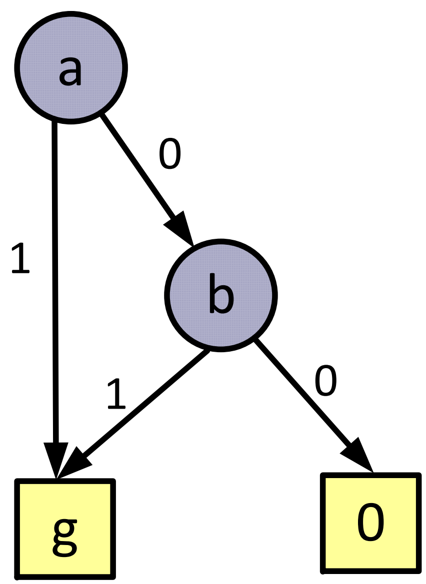


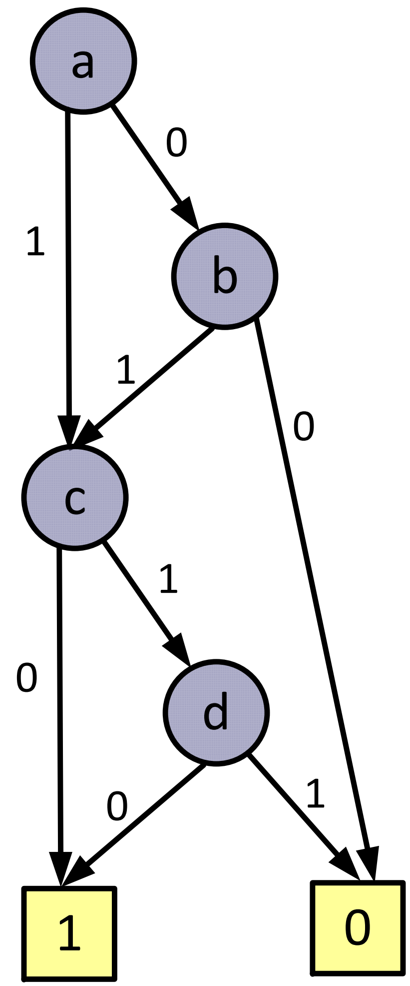
$$g = \overline{cd}$$



BDD of (f . g)

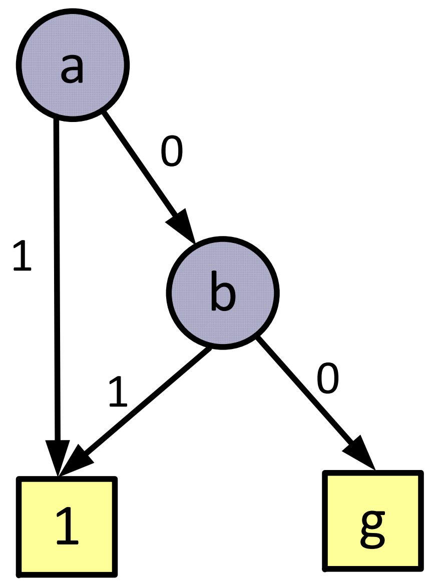
- BDD of h = f.g?

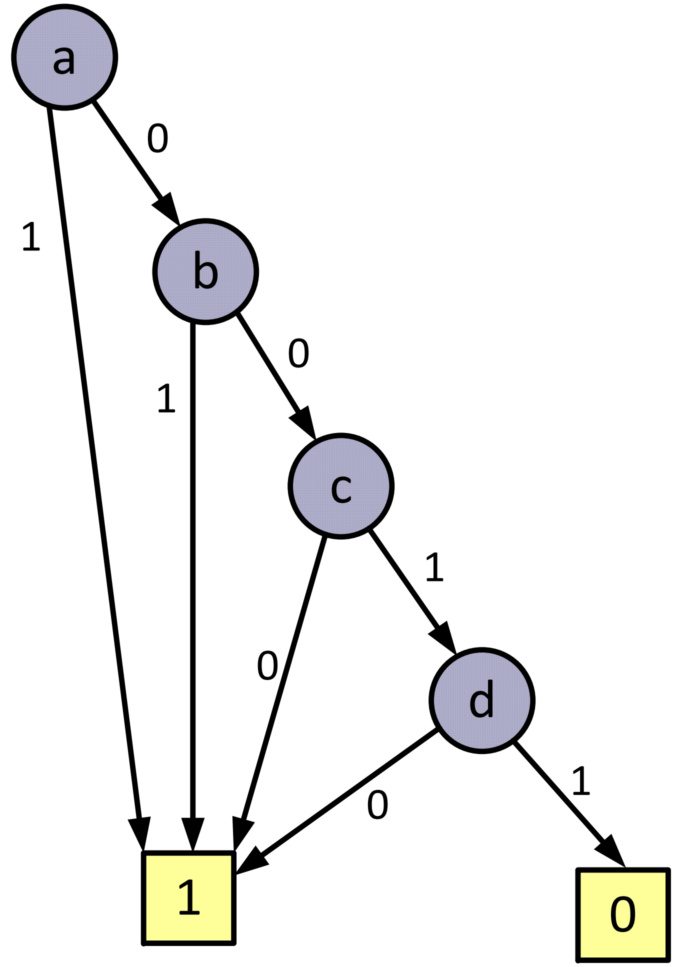




BDD of (f + g)

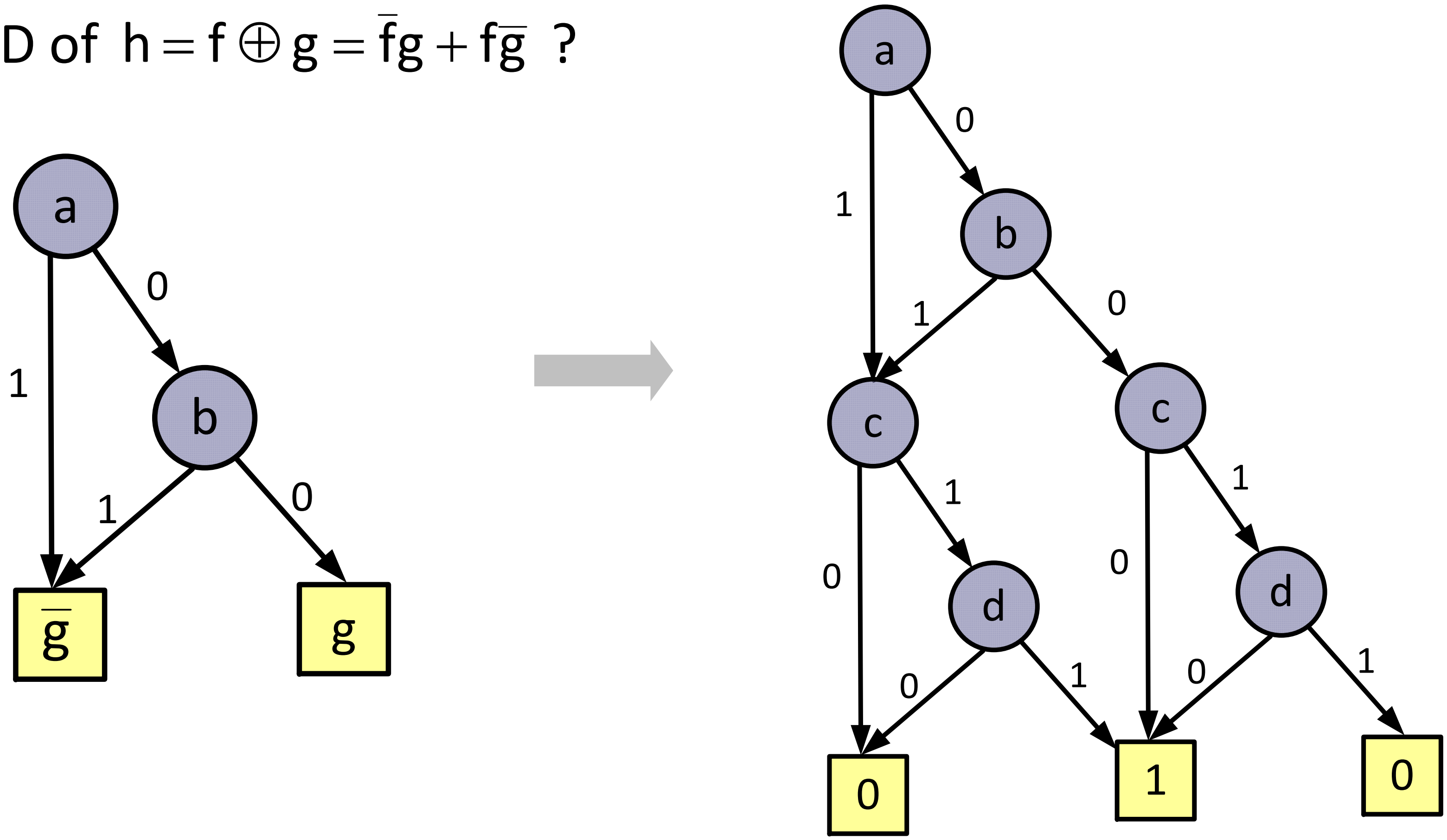
- BDD of h = f+g?





BDD of ($h = f \oplus g$)

- BDD of $h = f \oplus g = \bar{f}g + f\bar{g}$?

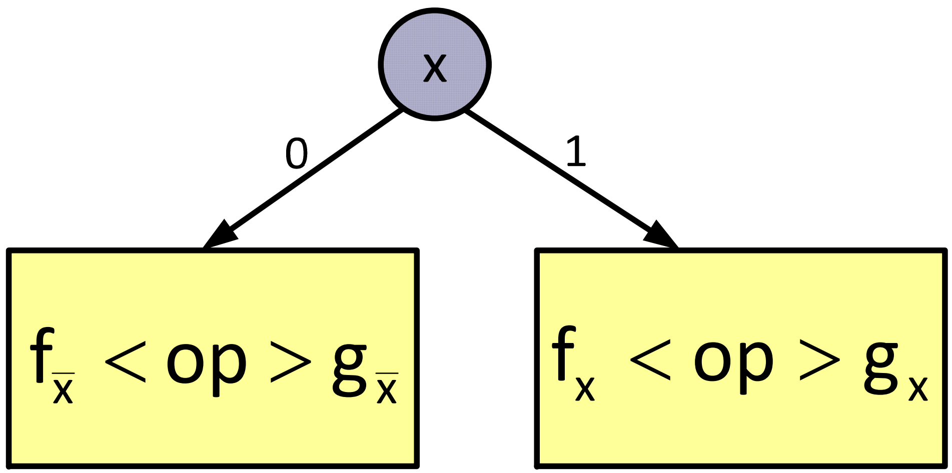


When the support of “f” and the support of “g” are disjoint, then replace terminals of f with 0, 1, g, or $\bar{g}$

BDD of ($h = f \oplus g$)

- **Case 2:** both functions have the same support (i.e., $sup(f) = sup(g)$)

$$h = f(x,y) < op > g(x,y) = \left[x f(1,y) + \bar{x} f(0,y)\right] < op > \left[x g(1,y) + \bar{x} g(0,y)\right]$$
$$= x\left[f(1,y) < op > g(1,y)\right] + \bar{x}\left[f(0,y) < op > g(0,y)\right]$$

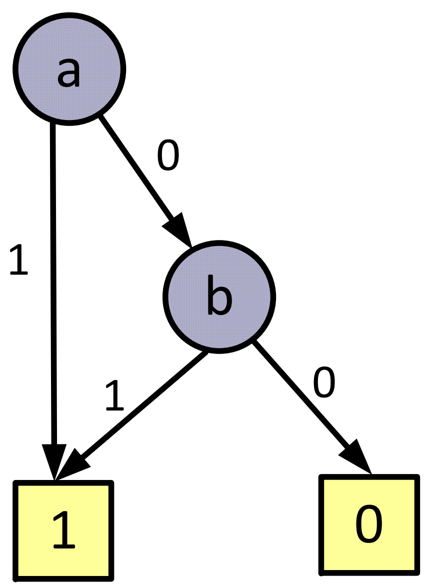


Pick a variable x and decompose both f and g with respect to it.
Then the 0-cofactor is $f_{\bar{x}} < op > g_{\bar{x}}$ and the 1-cofactor is $f_x < op > g_x$

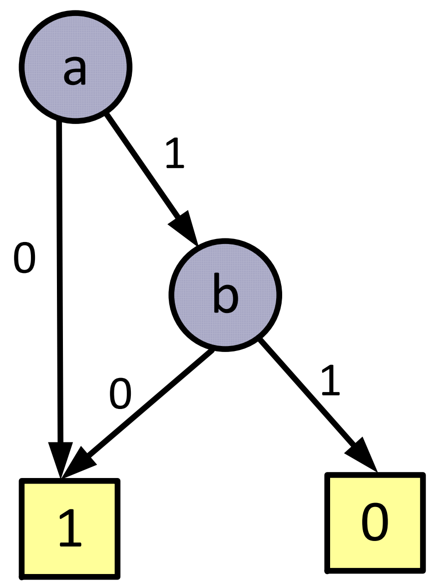
BDD of f <op> g

❖ **Example:** find f.g?

$f = a + b$

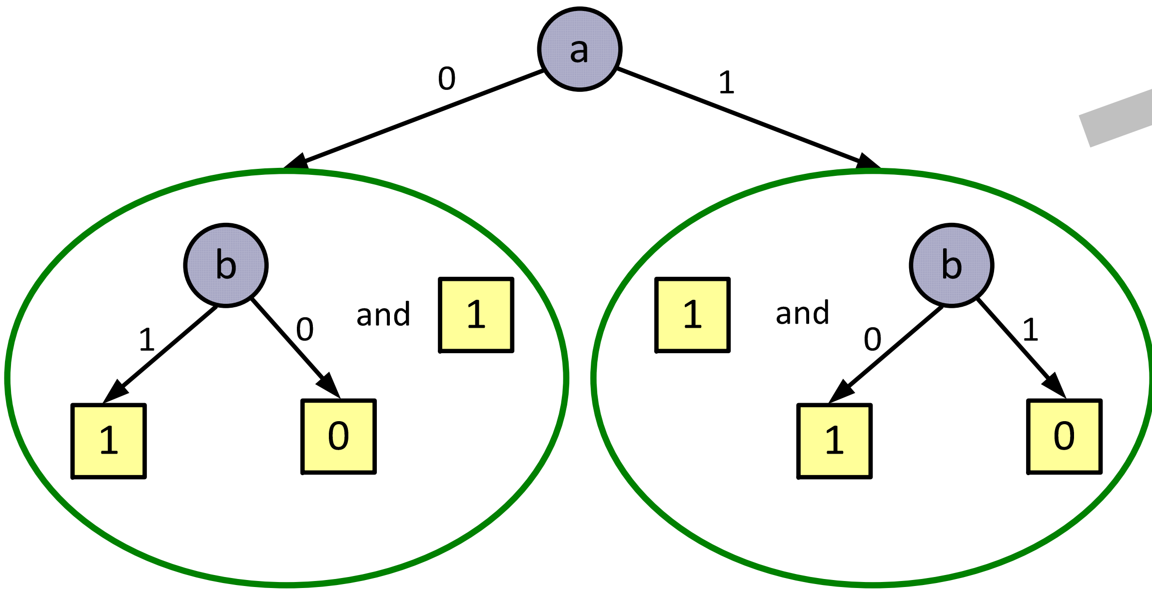


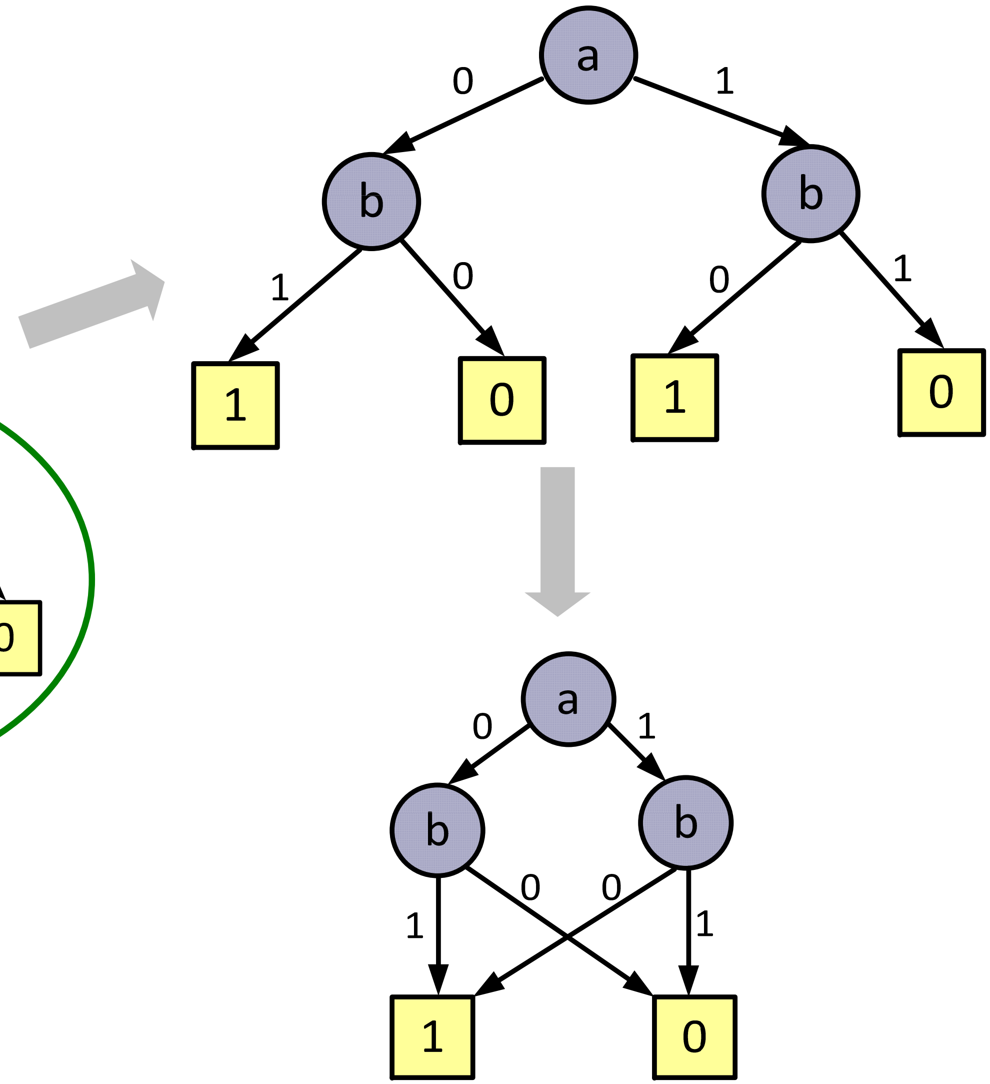
$g = \overline{ab}$



BDD of f . g

❖ **Example:** find f.g?





Boolean Satisfiability: Theory

❑ Boolean Satisfiability: (a.k.a SAT Problem)

- A Boolean expression in a conjunctive normal form (CNF):

$$f(x_1, x_2, \ldots, x_n) = (x_1 + x_2 + x_3)(x_4 + \overline{x}_5 + x_6)(\overline{x}_7 + \overline{x}_1 + x_8)\cdots(\cdots)$$

is satisfiable if and only if there exists a valuation for variables (x_1, x_2, …, x_n) such that f=1.

❖ **Example:** Is the following function satisfiable?

$$f = (x_1 + x_2)\ (x_1 + x_3)\ (\overline{x}_1 + x_3)(\overline{x}_1 + \overline{x}_3)$$

Try: x_1=1 ⇨ 1, 1, x_3=1, x_3=0 ✗ Conflict!

Try: x_1=0 ⇨ x_2=1, x_3=1, 1, 1 ✓ Satisfiable!

“f” is satisfiable.

Boolean Satisfiability: Theory

- **2-SAT:**
  - Boolean satisfiability problem where each sum term consists of no more than 2 literals
- **3-SAT:**
  - Boolean satisfiability problem where each sum term consists of no more than 3 literals
- If there are n variables in "f", (i.e., (x_1, x_2, …, x_n)), we have to search over 2^n possible cases to verify satisfiability!

NP-complete problem, i.e., non-deterministic polynomial time complete

- Is there any better way?
  - **YES** using dynamic programming algorithm based on Implication graphs.

Boolean Satisfiability: Implication Graph

❑ Implication Graph:

- An implication graph consists of
  - **Nodes:** which represent a variable and its assignments
  - **Edges:** which indicate an implication

❖ Example:

$$f = (\overline{a} + b + c)(a + c + d)(a + c + \overline{d})(a + \overline{c} + d)(a + \overline{c} + \overline{d})(\overline{b} + \overline{c} + d)(\overline{a} + b + \overline{c})(\overline{a} + \overline{b} + c)$$

$$\checkmark \quad \checkmark \quad \checkmark \quad \checkmark \quad \checkmark \quad \checkmark \quad \checkmark \quad \checkmark$$

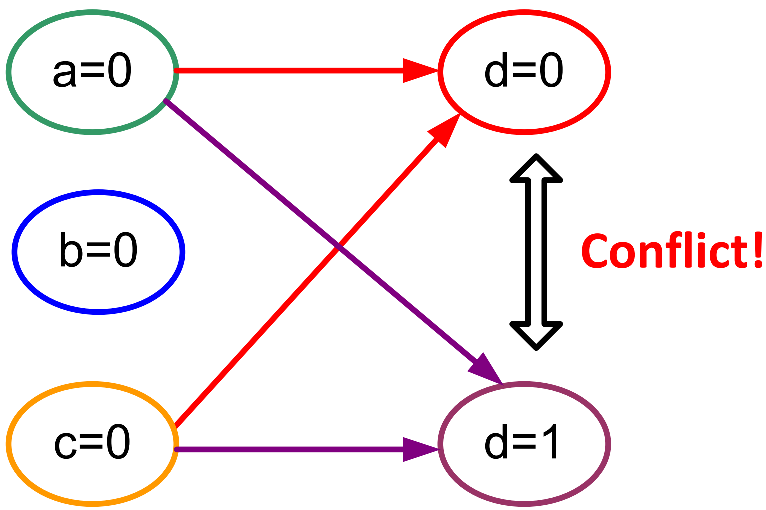
Satisfiable!
(SAT)

Boolean Satisfiability: Implication Graph

- **Implication Graph:**

$$f=(\overline{a}+b+c)(a+c+d)(a+c+\overline{d})(a+\overline{c}+d)(a+\overline{c}+\overline{d})(\overline{b}+\overline{c}+d)(\overline{a}+b+\overline{c})(\overline{a}+\overline{b}+c)$$

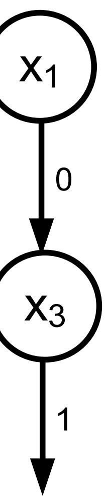
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓



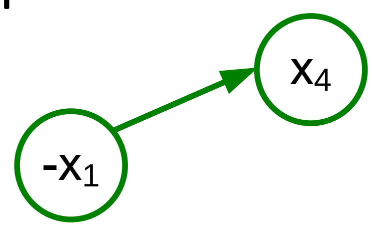
Boolean Satisfiability: Implication Graph

$$f=(x_1+x_4)(x_1+\overline{x}_3+\overline{x}_8)(x_1+x_8+x_{12})(x_2+x_{11})(\overline{x}_7+\overline{x}_3+x_9)(\overline{x}_7+x_8+\overline{x}_9)(x_7+x_8+\overline{x}_{10})(x_7+x_{10}+\overline{x}_{12})$$

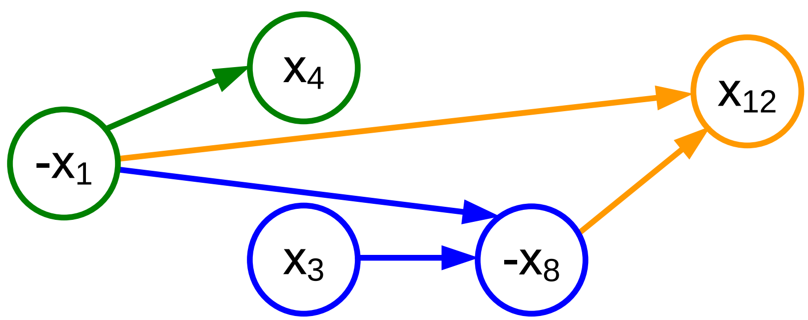
- **Idea:**
  - Assign a variable such that it simplifies the largest number of clauses (removes literal from a clause)
  - Draw an implication graph



- **Step 1**: $x_1=0 \Rightarrow x_4=1$



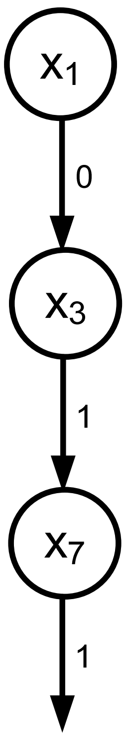
- **Step 2**: $x_3=1 \Rightarrow x_8=0$

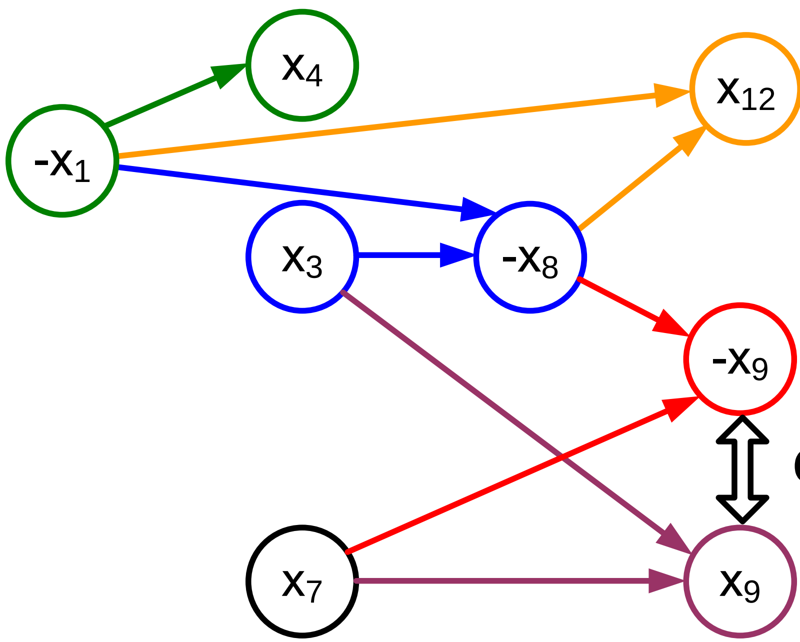


Boolean Satisfiability: Implication Graph

$$f = (x_1 + x_4)(x_1 + \bar{x}_3 + \bar{x}_8)(x_1 + x_8 + x_{12})(x_2 + x_{11})(\bar{x}_7 + \bar{x}_3 + x_9)(\bar{x}_7 + x_8 + \bar{x}_9)(x_7 + x_8 + \bar{x}_{10})(x_7 + x_{10} + \bar{x}_{12})$$

- **Step 3**: $x_7 = 1$





Conflict : $x_7 x_3 \bar{x}_8$

$(x_1 + x_4)$

$(x_1 + \bar{x}_3 + \bar{x}_8)$

$(x_1 + x_8 + x_{12})$

$(x_2 + x_{11})$

$(\bar{x}_7 + \bar{x}_3 + x_9)$

$(\bar{x}_7 + x_8 + \bar{x}_9)$

$(x_7 + x_8 + \bar{x}_{10})$

$(x_7 + x_{10} + \bar{x}_{12})$

Conflict Clause: $\overline{(x_7 x_3 \bar{x}_8)} = (\bar{x}_7 + \bar{x}_3 + x_8)$

This will be 1 when
$x_7 \neq 1$ or $x_3 \neq 1$ or $x_8 \neq 0$
So does not change the function

Boolean Satisfiability: Implication Graph

- When a conflict clause is found:
  - Add the conflict clause to f
  - Does not the conflict to happen again
  - Backtrack to the point where we first assigned one of the variables in the conflict clause
- So we should go back to step 2 where x_3=1:

✓ $(x_1 + x_4)$

✓ $(x_1 + \bar{x}_3 + \bar{x}_8)$

✓ $(x_1 + x_8 + x_{12})$

$(x_2 + x_{11})$

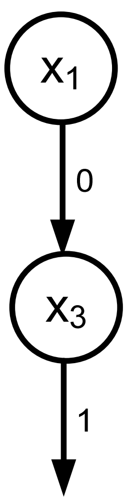
$(\bar{x}_7 + \bar{x}_3 + x_9)$

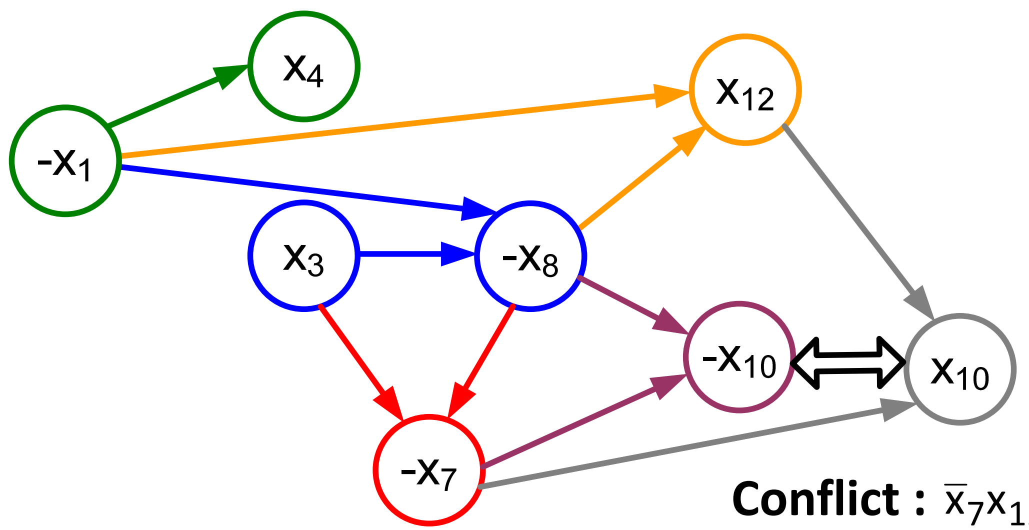
$(\bar{x}_7 + x_8 + \bar{x}_9)$

✓ $(x_7 + x_8 + \bar{x}_{10})$

✓ $(x_7 + x_{10} + \bar{x}_{12})$

✓ $(\bar{x}_7 + x_8 + \bar{x}_3)$





Conflict : $\bar{x}_7 x_{12} \bar{x}_8$

Conflict Clause: $x_7 + \bar{x}_{12} + x_8$

Boolean Satisfiability: Implication Graph

- Note: no variable that was previously set is present in the conflict clause. Therefore, the current branch is unsatisfiable, so explore the other branch, i.e., x_3=0.
- Also set x_7=0, which does not force anything
- So set x_8=0 then conflict!

✓ $(x_1 + x_4)$

✓ $(x_1 + \bar{x}_3 + \bar{x}_8)$

✓ $(x_1 + x_8 + x_{12}$

$(x_2 + x_{11})$

✓ $(\bar{x}_7 + \bar{x}_3 + x_9)$

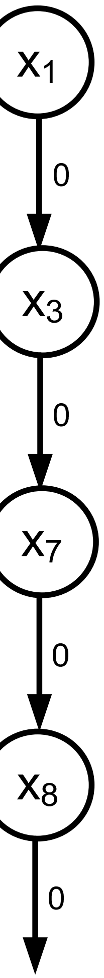
✓ $(\bar{x}_7 + x_8 + \bar{x}_9)$

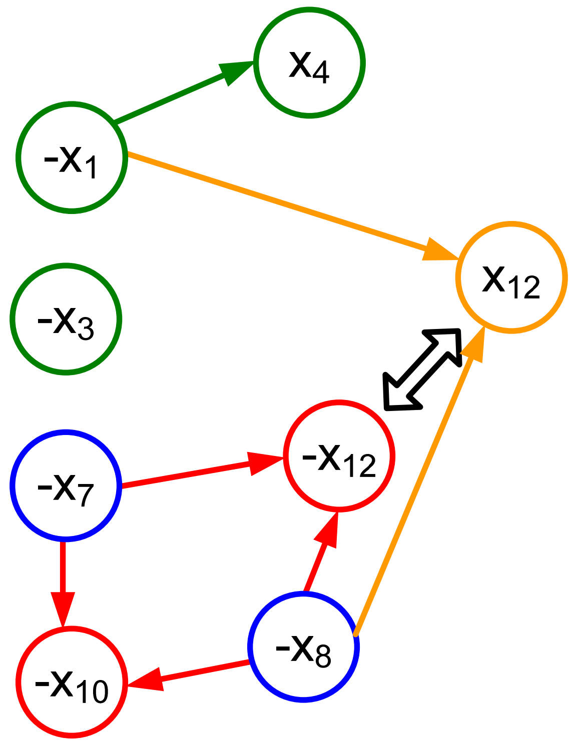
✓ $(x_7 + x_8 + \bar{x}_{10}$

$(x_7 + x_{10} + \bar{x}_{12}$

✓ $(\bar{x}_7 + x_8 + \bar{x}_3)$

✓ $(x_7 + x_8 + \bar{x}_{12}$

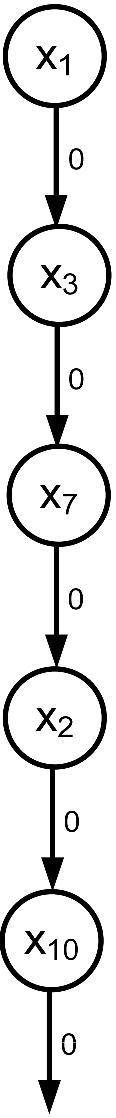




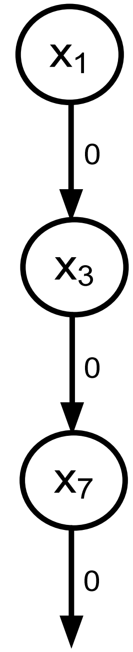
Conflict : $\bar{x}_7\bar{x}_1\bar{x}_8$

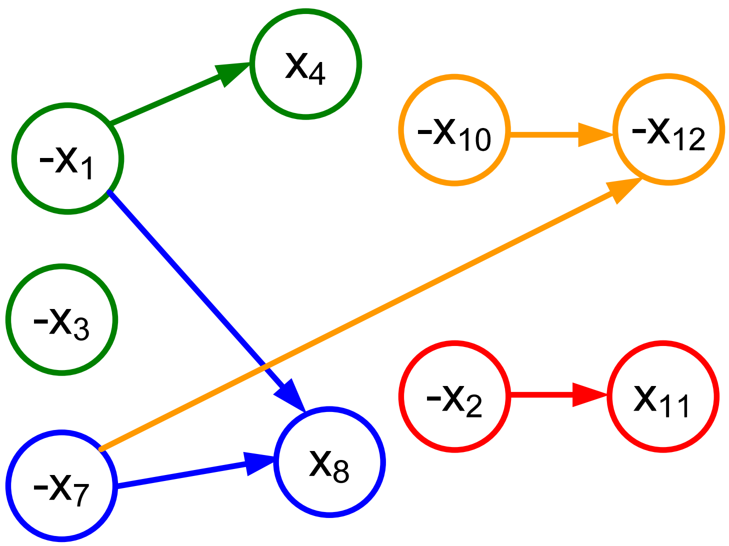
Conflict Clause: $x_7 + x_1 + x_8$

Boolean Satisfiability: Implication Graph



- Go back to x_1=0, x_3=0, x_7=0.
- Set x_2=0
- Set x_{10}=0





✓ $(x_1 + x_4)$
✓ $(x_1 + \bar{x}_3 + \bar{x}_8)$
✓ $(x_1 + x_8 + x_{12})$
✓ $(x_2 + x_{11})$
✓ $(\bar{x}_7 + \bar{x}_3 + x_9)$
✓ $(\bar{x}_7 + x_8 + \bar{x}_9)$
✓ $(x_7 + x_8 + \bar{x}_{10})$
✓ $(x_7 + x_{10} + \bar{x}_{12})$

- - - - - - - - - - -

✓ $(\bar{x}_7 + x_8 + \bar{x}_3)$
✓ $(x_7 + x_8 + \bar{x}_{12})$
✓ $(x_7 + x_8 + x_1)$

Satisfiable! (SAT)

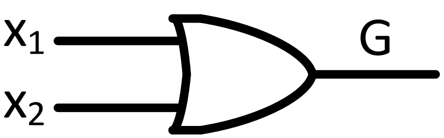
We explored 12 cases of 2^{12}=4096 possible cases!

Boolean Satisfiability: SAT Solvers

- Go back to our question.
- Given a logic structure can we implement a given logic expression within it?

- **Characteristic Equation (CE):**
  - A logical expression that takes as inputs all literals of a function as well as its output, and it produces a "1" iff for a given set of inputs the output is correct

❖ Example: Dose G = f ?



x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	G	F_{OR}
0	0	0	1
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

G follows the OR-gate behavior

Means G should not be "0" when x_1=1 & x_2=0

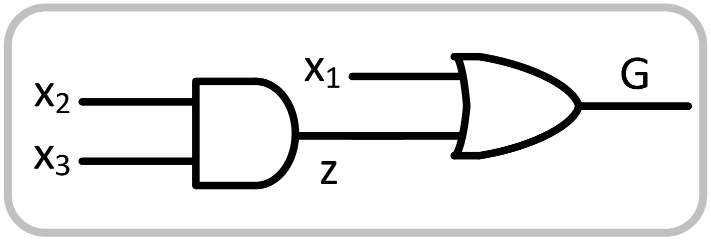
Characteristic Equation

$$F = (\overline{x}_2 + G)(\overline{x}_1 + G)(x_1 + x_2 + \overline{G})$$

If we substitute (x_1+x_2) for G then F=1
Thus, F is SAT so G implements an OR function

Boolean Satisfiability: SAT Solvers

- **Characteristic Equation (CE) for a Logic Network:**
- **Divide-and-Conquer**
  - Create a CE for each gate
  - Combine CEs for gates to form a CE for the network



x_2	x_3	z	F_{AND}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$F_{OR} = (\overline{z} + G)(\overline{x}_1 + G)(x_1 + z + \overline{G})$$

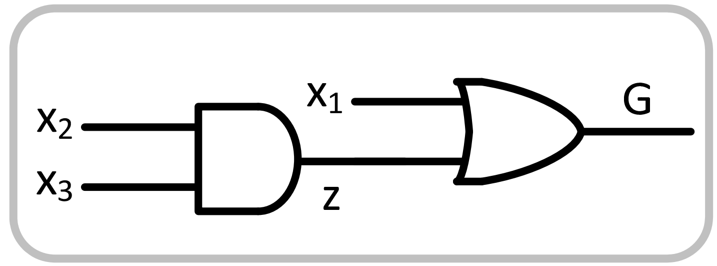
$$F_{AND} = (x_3 + \overline{z})(x_2 + \overline{z})(\overline{x}_3 + \overline{x}_2 + z)$$

$$\Downarrow$$

$$F = F_{OR} \cdot F_{AND} = (x_3 + \overline{z})(x_2 + \overline{z})(\overline{x}_3 + \overline{x}_2 + z)(\overline{z} + G)(\overline{x}_1 + G)(x_1 + z + \overline{G})$$

Boolean Satisfiability: SAT Solvers

- **SAT Theorem:**
  - In order for a given logic network, represented by a characteristic equation F, to be equivalent to a logic function H, then for all x_is, F(G=H) must be SAT.



❖ **Example:** Assume $H = x_1+x_2$, find out if G = H ?

- This question is equivalent to see if $F(G= x_1+x_2)$ is SAT for all $x_1,x_2\ ,x_3$ values?

$$F\left(G\big|_{x_1+x_2}\right) = (x_3+\bar{z})(x_2+\bar{z})(\bar{x}_3+\bar{x}_2+z)(\bar{z}+x_1+x_2)(\bar{x}_1+(x_1+x_2))(x_1+z+\overline{(x_1+x_2)})$$

$$x_3 = 0 \;\Longrightarrow\; F = x_1 + \bar{x}_1\bar{x}_2$$ (Not SAT for all x_1, x_2 values)

So we can NOT implement an OR function in F

SAT Theorem

- **SAT Theorem:**

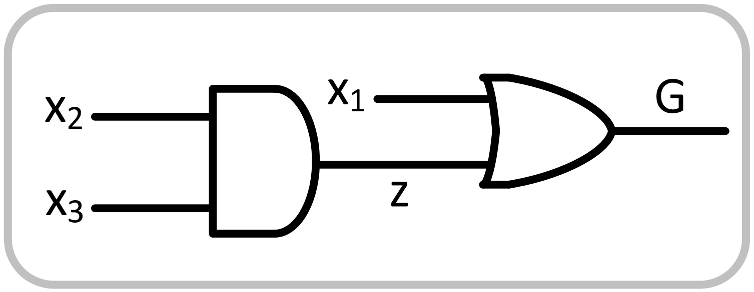
Characteristic Equation **F** represents **H** if and only if:

$$\forall i, \forall j, \forall x_i \; \exists z_j \; F(G = H) : SAT$$

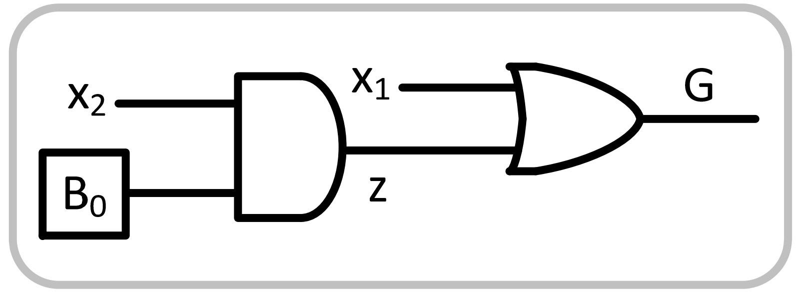
- x_i : Literals
- z_j : Internal wires

Quantified SAT (QSAT) Problem:

SAT Problem



QSAT Problem



Question:

Can **F(G)** implement **H** for all x_1,x_2 ,x_3 values?

Question:

Can **F(G)** implement $\mathbf{H=x_1+x_2}$ for all x_1,x_2 values by adjusting B_0?

Quantified SAT (QSAT) Problem:

- **Step 1**: Generate CE for F(G):

$$F = (B_0 + \bar{z})(x_2 + \bar{z})(\overline{B_0} + \bar{x}_2 + z)(\bar{z} + G)(\bar{x}_1 + G)(x_1 + z + \overline{G})$$

- **Step 2**: For each valuation of x_1, x_2, determine what B_0 should be in order for G to implement H

➢ i) $F_{\bar{x}_1\bar{x}_2}(G) = (B_0 + \bar{z})(\bar{z})(1)(\bar{z} + G)(1)(z + \overline{G})$

$H(x_1 = 0, x_2 = 0) = 0 \Rightarrow G = 0 \Rightarrow z = 0 \Rightarrow \boxed{B_0 : \text{don't care}}$

➢ ii) $F_{\bar{x}_1 x_2} = (B_0 + \bar{z})(1)(\overline{B_0} + z)(\bar{z} + G)(1)(z + \overline{G})$

$H(x_1 = 0, x_2 = 1) = 1 \Rightarrow G = 1 \Rightarrow z = 1 \Rightarrow B_0 = 1$

Quantified SAT (QSAT) Problem:

Step 2, Cont'd:

- iii) $F_{x_1\bar{x}_2} = (B_0 + \bar{z})(\bar{z})(1)(\bar{z} + G)(G)(1)$

  $H(x_1 = 1, x_2 = 0) = 1 \Rightarrow G = 1 \Rightarrow z = 0 \Rightarrow$ $\boxed{B_0 : \text{don't care}}$

- iv) $F_{x_1x_2}(G) = (B_0 + \bar{z})(\overline{B_0} + z)(1)(\bar{z} + G)(G)(1)$

  $H(x_1 = 1, x_2 = 1) = 1 \Rightarrow G = 1 \Rightarrow z = 1 \Rightarrow B_0 = 1 \Rightarrow$ $\boxed{\text{SAT}}$

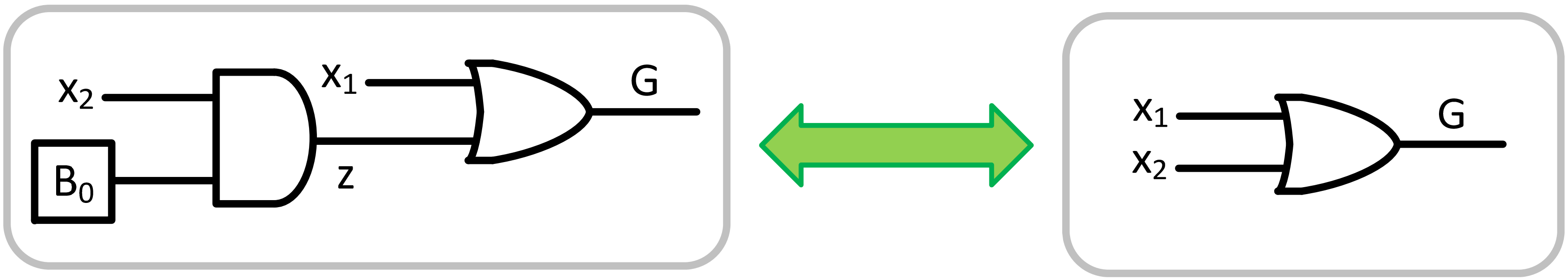
  $G = 1 \Rightarrow z = 0 \Rightarrow B_0 = 0$

  $\boxed{\text{Contradicts the results from } F_{\bar{x}_1x_2}}$

All Fs are SAT and B_0=1 is the consistent solution

Quantified SAT (QSAT) Problem:

QSAT Problem



Question:

Can **F(G)** implement **$H=x_1+x_2$** for all x_1,x_2 values by adjusting B_0?

Answer:

YES for B_0=1

Boolean Satisfiability: SAT Solvers

- Can we solve this problem with a single equation?
  - Yes, solve SAT for

$$F = F_{\bar{x}_1\bar{x}_2} \cdot F_{\bar{x}_1 x_2} \cdot F_{x_1\bar{x}_2} \cdot F_{x_1 x_2}$$

  - G is replaced with $H(x_1, x_2)$
  - Note that signal z should be replicated as
    - z_1 for $F_{\bar{x}_1\bar{x}_2}$
    - z_2 for $F_{\bar{x}_1 x_2}$
    - z_3 for $F_{x_1\bar{x}_2}$
    - z_4 for $F_{x_1 x_2}$
  - This is b/c z is an intermediate variable and based on the above calculations z results in a conflict
  - The choice of B_0=1 will be resolved due to conflict clauses

QSAT Theorem

- **QSAT Theorem:**

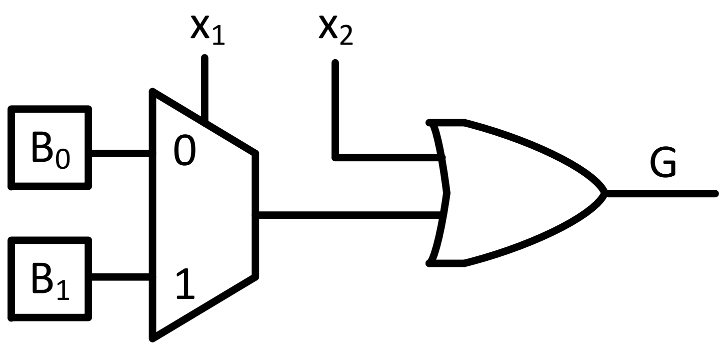
Characteristic Equation **F** represents **H** if and only if:

$$\forall i, \forall j, \forall k, \exists B_k \forall x_i \, \exists z_j \, F(G = H) : SAT$$

- x_i : Literals
- z_j : Internal wires

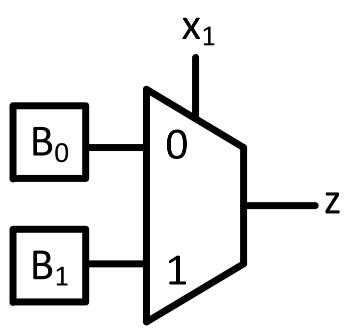
QSAT Theorem

- Can we implement function $H = x_1+x_2$ into:



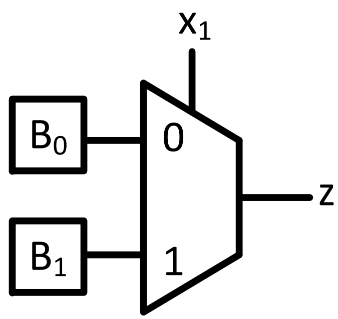
- We are looking for a value for B_0 and a value for B_1

- **Step 1**: Represent G as a logic expression. This expression is the characteristic equation



QSAT Theorem

- **Step 1**: Represent MUX as a logic expression. This expression is the characteristic equation of the MUX



I) $B_0 + x_1 + \overline{z}$

II) $B_1 + \overline{x}_1 + \overline{z}$

III) $\overline{B_0} + x_1 + z$

IV) $\overline{B_1} + \overline{x}_1 + z$

B_0	B_1	x_1	z	F_{MUX}
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

I

II

IV

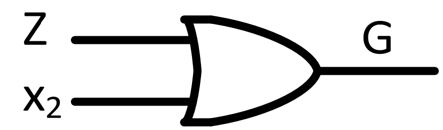
III

$$F_{MUX} = (\overline{B_0} + x_1 + \overline{z})(B_1 + \overline{x}_1 + z)(\overline{B_0} + x_1 + z)(\overline{B_1} + \overline{x}_1 + z)$$

Boolean Satisfiability: SAT Solvers

$$F_{OR} = (\overline{x}_2 + G)(\overline{z} + G)(z + x_2 + \overline{G})$$

x_2	Z	G	F_{OR}
0	0	0	1
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



- Thus the complete characteristic equation for the logic structure is:

$$F = (\overline{B}_0 + x_1 + \overline{z})(B_1 + \overline{x}_1 + z)(\overline{B}_0 + x_1 + z)(\overline{B}_1 + \overline{x}_1 + z)(x_2 + z + \overline{G})(\overline{x}_2 + G)(\overline{z} + G)$$

- **Step2:** Take F and see if a given logic expression can be implemented in it.

This means that we need to find a value for B_0 and B_1 such that for all x_1, x_2 values, B_0 and B_1 are consistent.

Quantified SAT (QSAT) Problem:

- **Step 2**: For each valuation of x_1, x_2, determine what B_0 and B_1 should be in order for G to implement $H = x_1+x_2$

  - i) $F_{\bar{x}_1\bar{x}_2}(G) = (B_0 + \bar{z}_1)(1)(\overline{B}_0 + z_1)(1)(z_1 + \overline{G})(1)(\bar{z}_1 + G)$

    G has to be equal to H, i.e., 0

    $\boxed{G = 0 \Rightarrow z_1 = 0 \Rightarrow B_0 = 0 \quad H = 0 \ \checkmark}$

  - ii) $F_{x_1\bar{x}_2} = (1)(B_1 + \bar{z}_2)(1)(\overline{B}_1 + z_2)(z_2 + \overline{G})(1)(\bar{z}_2 + G)$

    G has to be equal to H, i.e., 1

    $\boxed{G = 1 \Rightarrow z_2 = 1 \Rightarrow B_1 = 1 \quad H = 1 \ \checkmark}$

Quantified SAT (QSAT) Problem:

- iii) $F_{\bar{x}_1 x_2}(G) = (B_0 + \bar{z}_3)(1)(\bar{B}_0 + z_3)(1)(1)(G)(\bar{z}_3 + G)$

$$H = 1 \Rightarrow G = 1 \Rightarrow \boxed{z_3 = 0 \Rightarrow B_0 = 0 \ \checkmark}$$

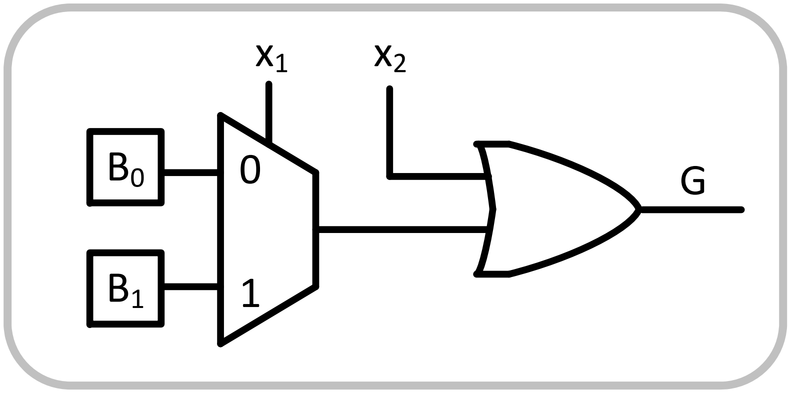
- iv) $F_{x_1 x_2} = (1)(B_1 + \bar{z}_4)(1)(\bar{B}_1 + z_4)(1)(G)(\bar{z}_4 + G)$

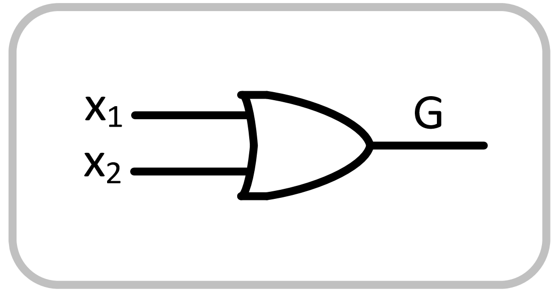
$$H = 1 \Rightarrow G = 1 \Rightarrow \begin{cases} \boxed{B_1 = 1 \Rightarrow z_4 = 1 \ \checkmark} \\ B_1 = 0 \Rightarrow z_4 = 0 \ \times \end{cases}$$

No surprise as when x_1=1 then z_4 has to be equal to B_1

Find a consistent value: B_1 = 1 and B_0 = 0

Quantified SAT (QSAT) Problem:





Question:

Can **F(G)** implement **$H=x_1+x_2$** for all x_1,x_2 values by adjusting B_0?

Answer:

YES for $B_0=0$ and $B_1=1$

Outline

- Introduction to Synthesis
- Digital Logic Basics
- Logic Optimization
  - Two-level logic synthesis
  - Multi-level logic synthesis
- **Technology Mapping**
  - Boolean Satisfiability
  - ASIC/FPGA-oriented Technology Mapping

Logic Synthesis

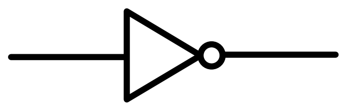
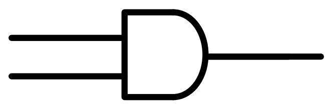
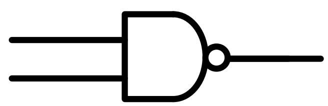
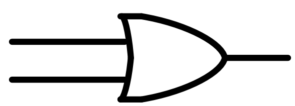
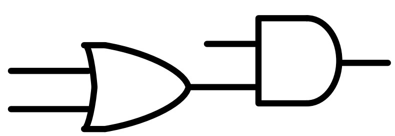
- Logic synthesis normally in two steps:
  - **Technology independent**
    - Manipulate equations
    - Optimize the logic equations
    - Independent of target IC media
  - **Technology dependent (Technology Mapping (TM))**
    - Equations are turned into netlist of the available gates

Technology Mapping

- **Problem Definition:**
  - **Given:**
    1. Boolean network G(v,e), where
       $v \in V$: represent logic functions
       $e \in E$: represent dependencies between logic functions
    2. Library of available gates
  - **Find:**
    - Netlist of gates from library that implements logic function G so as to minimize some of the following metrics:
      - o Area
      - o Delay
      - o Power
      - o Defect

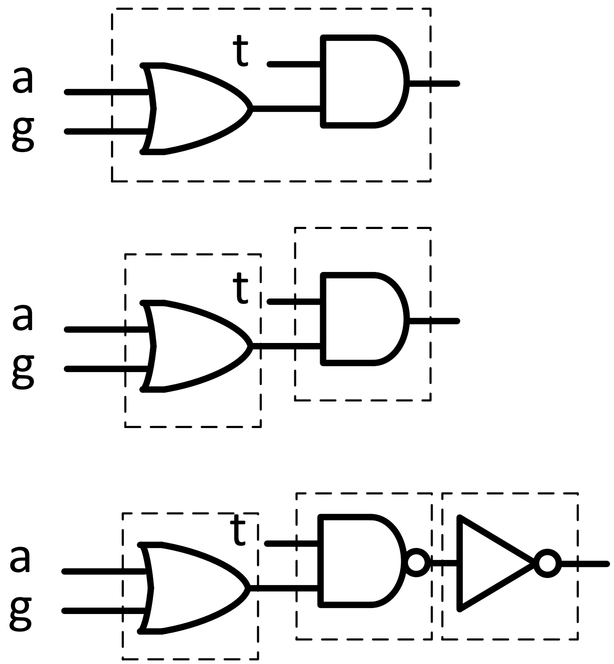
Technology Mapping (TM)

❖ Example: Implement function y=t(a+g) given the following library

Library Element		Area (sq units)	Delay (nsec)
INV		3	1 ns
AND		8	2.5 ns
NAND		6	2 ns
OR		8	2.5 ns
AND-OR		25	4 ns

Technology Mapping (TM)

- ❑ Possible Implementations:



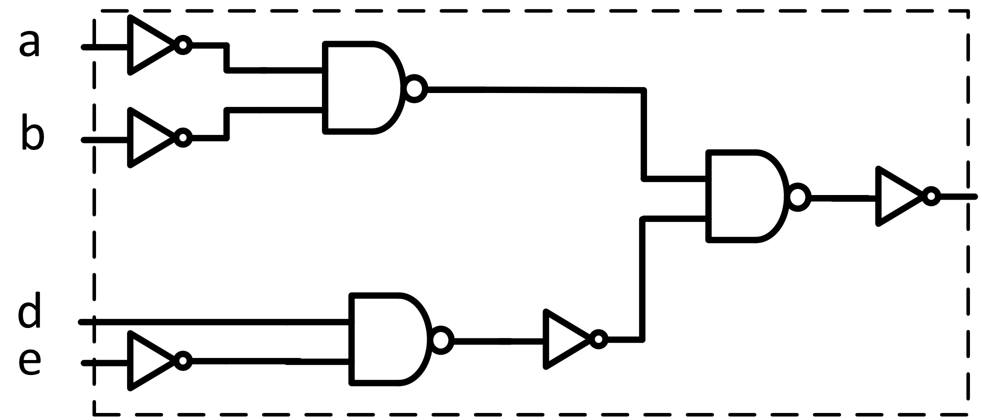
	Area	Delay	
1	25	4	**Best Delay**
2	16	5	**Best Area**
3	17	5.5	

- ❑ This is a small function with small library ➡ 3 choices
- ❑ Larger designs + libraries ➡ many more choices

Technology Mapping (TM)

- Algorithmic approach to TM (DAGON Keutzer 80s)
  1. Take input Boolean network and do a simple mapping into a network of "**base functions**" to create a "**subject graph"**
     e.g., 2-input NANDs + inverters

$$F = d\overline{e}(a+b)$$
$$= d\overline{e} \cdot \overline{(\overline{a}\,\overline{b})}$$

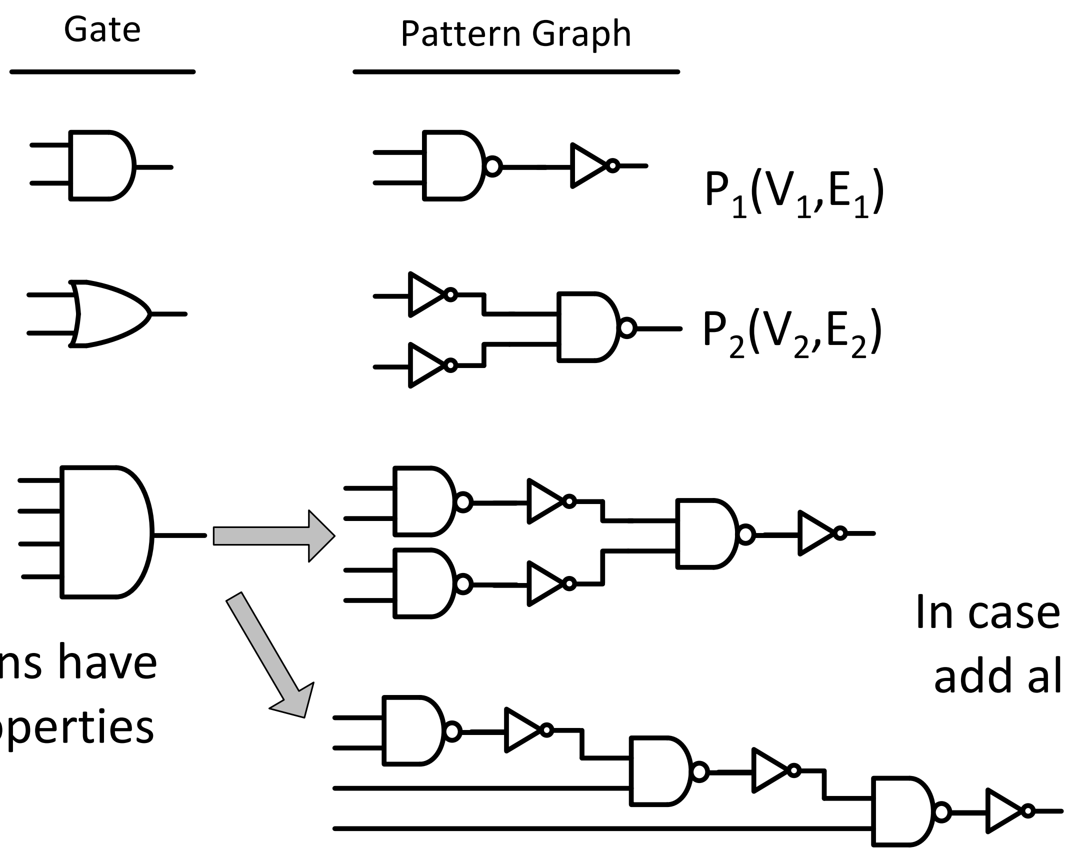


"Subject Network"

G(V,E)

Technology Mapping (TM)

- Algorithmic approach to TM (DAGON Keutzer 80s)
  2. Take gates in library and create a "**pattern graph**" for each that is functionally equivalent and expressed in the same base functions



Different implementations have different area/delay properties

In case there are alternatives, add all of the pattern graphs

Technology Mapping (TM)

- Now we have subject graph G(V,E) and set of pattern graphs $P_1(V_1,E_1)$, $P_2(V_2,E_2)$,…, $P_L(V_L,E_L)$ with their area and delay properties

- We have to cover G(V,E) with the set of pattern graphs that results in a minimum cost

- Cost: area (sum of area of all gates), delay (max delay along any path),…

- Find the minimum-cost cover is equivalent to dynamic programming (DP)
- **DP Idea:**
  - Break the overall problem into sub-problems
  - Solve sub-problems optimally (store results in a table)
  - Use solutions to sub-problems, construct solution to the overall problem

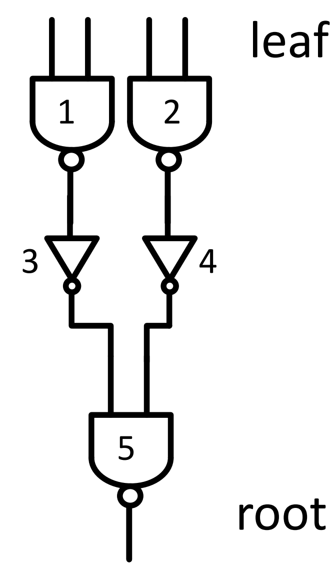
Technology Mapping (TM)

- **DP for TM: (Work Backward)**
  - Begin at leafs of tree and find optimal mapping of each leaf node
  - Move up tree find optimal mappings for sub-trees using already-computed mappings

$$
\begin{aligned}
F &= \bar{a} + \bar{b} + \bar{c} + \bar{d} \\
&= \overline{ab} + \overline{cd} \\
&= \overline{\overline{\overline{ab}} \cdot \overline{\overline{cd}}}
\end{aligned}
$$

Create subject graph using base functions

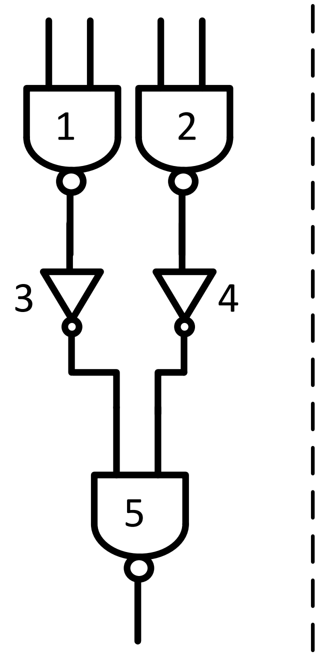
Subject Graph



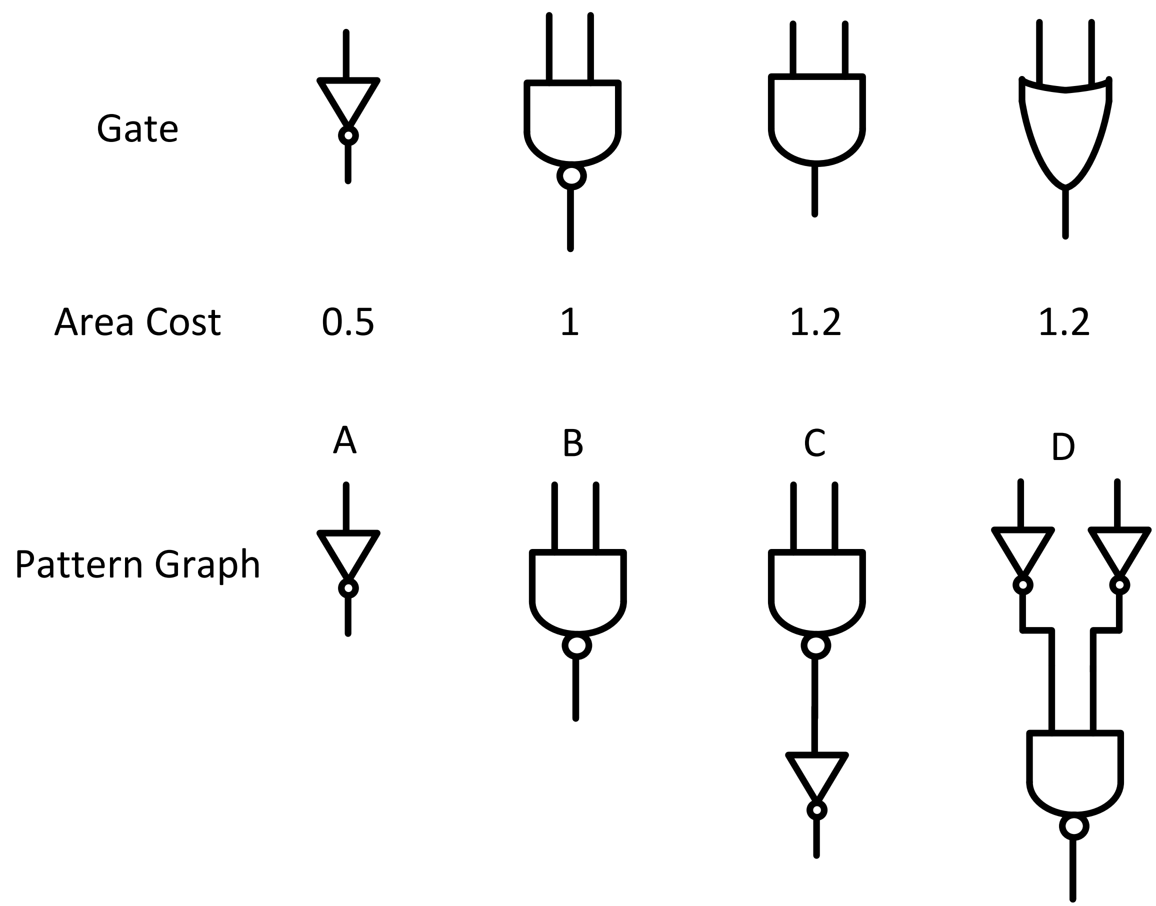
Technology Mapping (TM)

❑ DP for TM: (Work Backward)

Subject Graph



List of gates, costs, and pattern graphs



Technology Mapping (TM)

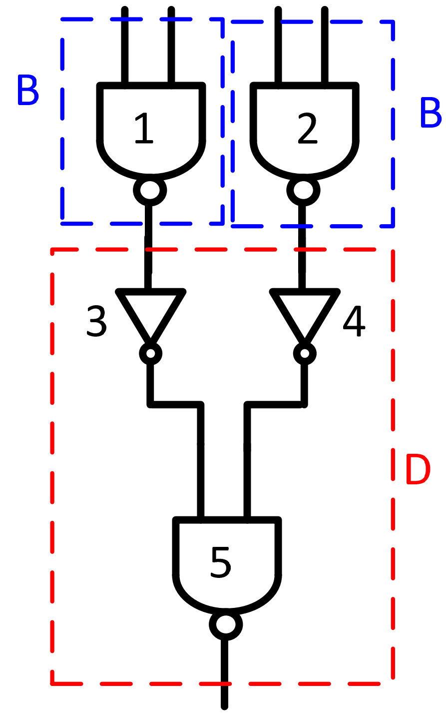
- Begin at leafs, traverse all nodes
  - Find all pattern graphs that match at each node (match the entire pattern graph all the way back (with trace back))
  - Record the best total cost to implement sub-tree rooted at node

1) B matches, cost = 1 ⟹ record this match + cost
2) B matches, cost = 1 ⟹ record this match + cost
3) A matches, cost = 0.5 +1 (cost shared for node 1)
   C matches, cost = 1.2 ⟹ record this match + cost
4) Same as node 3
5) B matches, cost = 1 + 1.2 + 1.2 =3.4
   D matches, cost = 1.2 + 1 + 1 = 3.2 ⟹ Best for node 5

Solution: D for node 5, B for nodes 1 and 2 (Optimal Mapping)

Technology Mapping (TM)

- **Optimal Solution:**



- **Key DP point:**
  - If we later need to implement the output of 3 explicitly (as the input to another gate), we need only consider using C for node 3 (DP result).

Technology Mapping (TM)

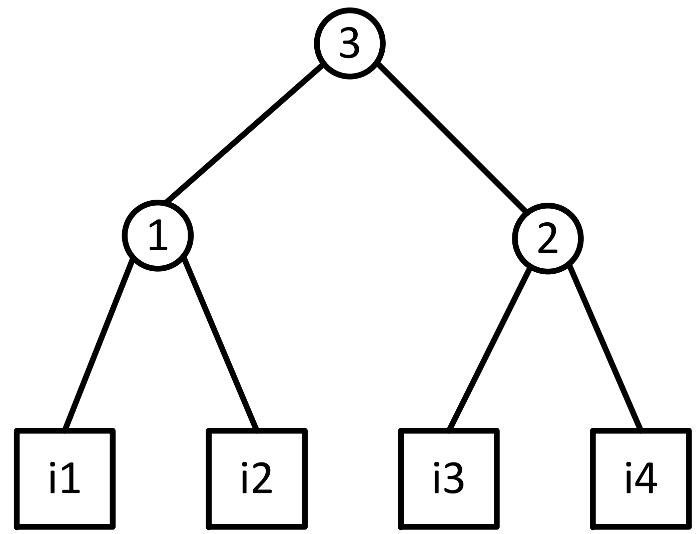
- **Summary:**
  - Post-order tree traversal (from leaves to the root)
  - For each node, find all the pattern graphs that match with trace back
  - Choose the one with the best cumulative cost
  - At root of tree, have optimal cost, trace back to construct mapping
  - Can also be used for delay (carry forward the longest path delay)

FPGA Technology Mapping (TM)

- **Target gates:** LUTs with K-inputs that can implement any function with K inputs
- Subject graph need not be in base functions
- Nodes must have less than K inputs (K-bounded network)
- Same bottom-up DP TM algorithm
  - Matching is different
  - Only need to care about # of inputs not functionality

FPGA Technology Mapping (TM)

- **Example:**
  - Mapping to 3-LUTs (K=3)



- What are matches at node 3?
  - Each match is a cut
  - Cuts of nodes 3:
    - {i1, i2, 2}
    - {i3, i4, 1}
    - {1,2}

- Finding all matches at a node = finding set of K-feasible cuts for that node

A cut is K-feasible if its size is less than K