Technology Mapping

Ref

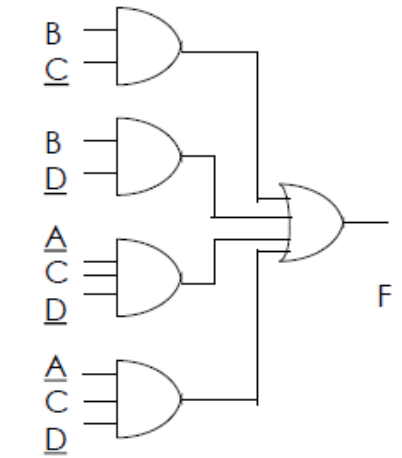
Synthesis and Optimization of Digital Circuits by G.D.Micheli

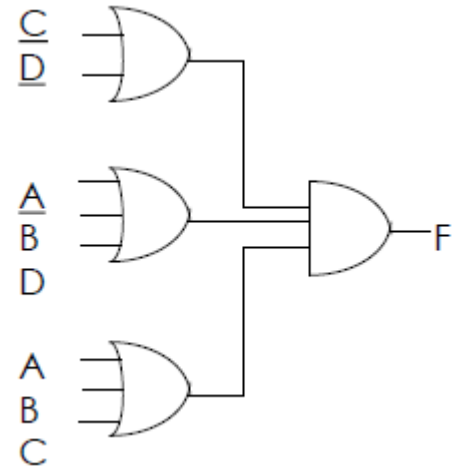
Introduction

- Objective: To relate the circuit representation to that of the cell library and to find a cell interconnection
- Classified into two major groups: ***heuristic*** algorithms and ***rule-based*** approaches
- Two-level & Multi-level logic representations
- Covering
  - Portion of a logic network can be replaced by a library cell
  - Selecting adequate number of instances of library elements to cover the logic network while optimizing figure of merit, such as area and/or delay
- Cell ***matches*** a subnetwork when they are functionally equivalent
- A cell may match a subnetwork even if the number of inputs differs and some of these are shorted together or connected to a fixed voltage rail

Two-Level Logic

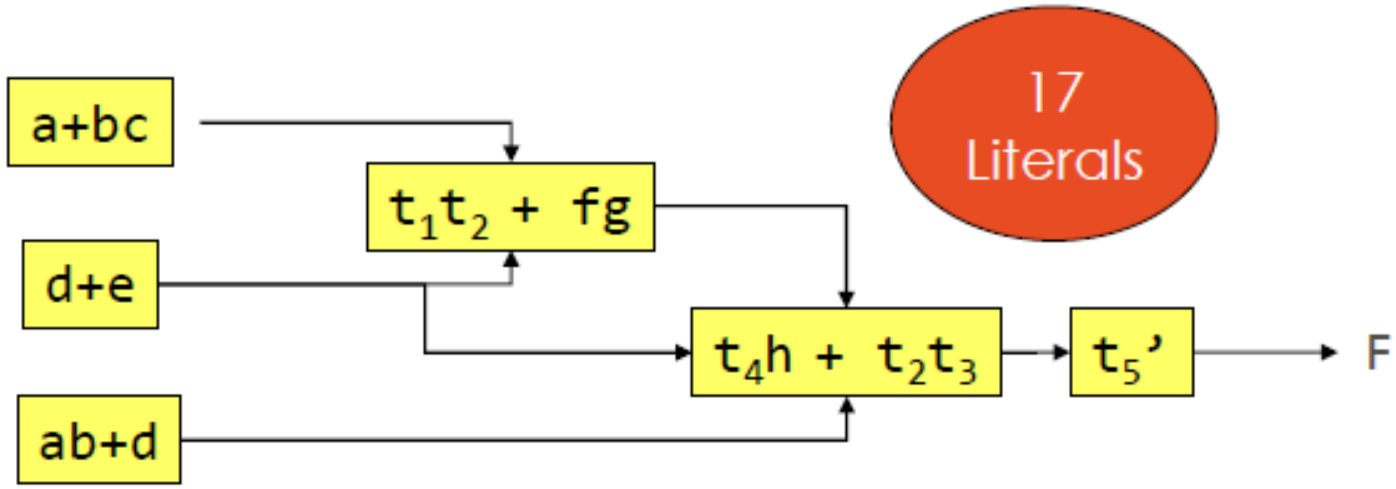
- **During elaboration, primary inputs and outputs (ports) are defined and sequential elements (flip-flops, latches) are *inferred***
- **This results in a set of combinational logic clouds with:**
- Input ports and register outputs are inputs to the logic
- Output ports and register inputs are the outputs of the logic
- The outputs can be described as Boolean functions of the inputs
- The goal of Boolean minimization is to reduce the number of literals in the output functions
- **Many different data structures are used to represent the Boolean functions:**
- Truth tables, cubes, Binary Decision Diagrams, equations, etc
- A lot of the research was developed upon SOP or POS representation, which is better known as “Two-Level Logic”





Multi-Level Logic

$t_1 = a + bc;$
$t_2 = d + e;$
$t_3 = ab + d;$
$t_4 = t_1t_2 + fg;$
$t_5 = t_4h + t_2t_3;$
$F = t_5';$



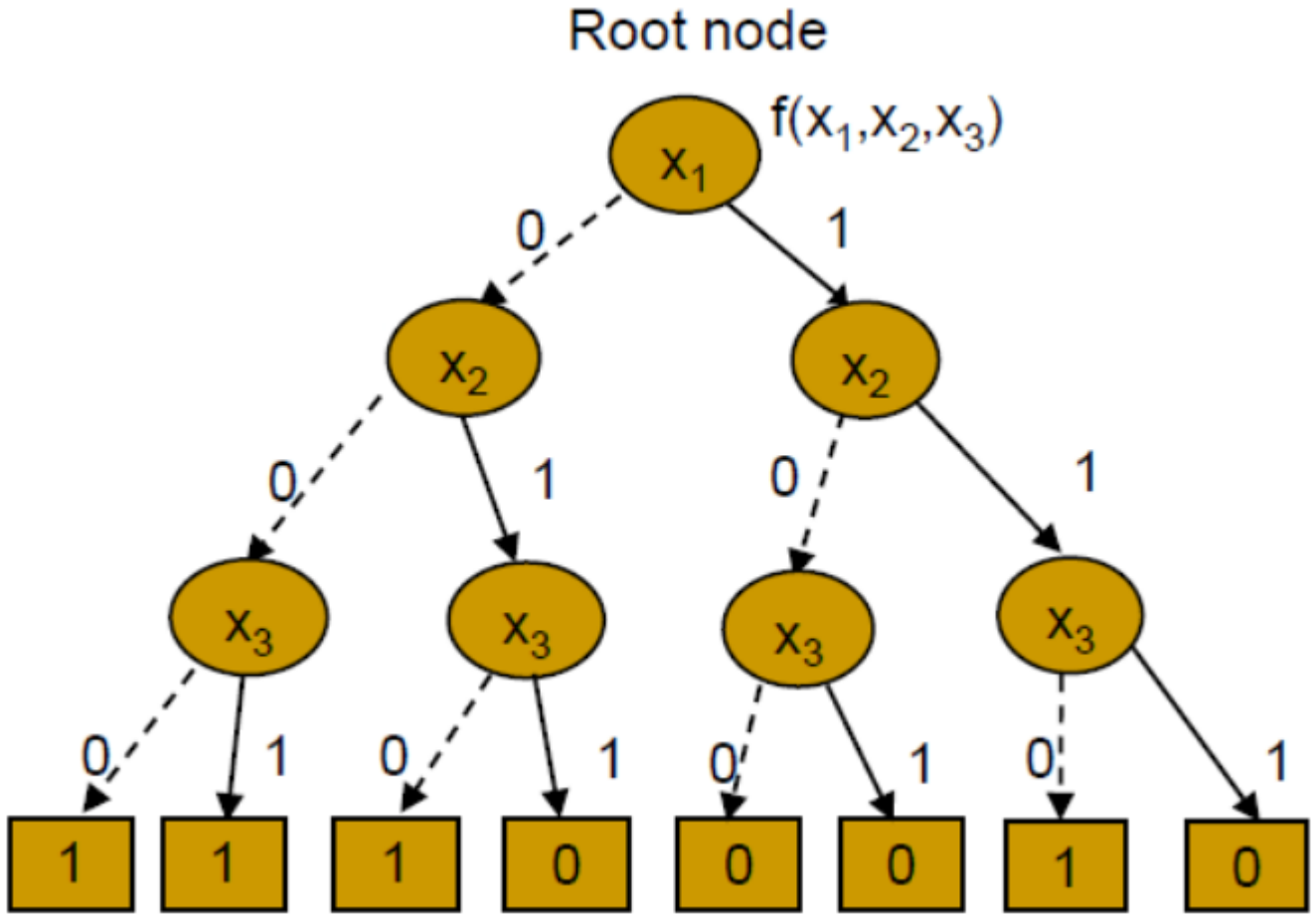
Minimize Multi-level Logic using various techniques to reduce number of variables

Binary Decision Diagrams (BDD)

- **BDDs are DAGs that represent the truth table of a given function**

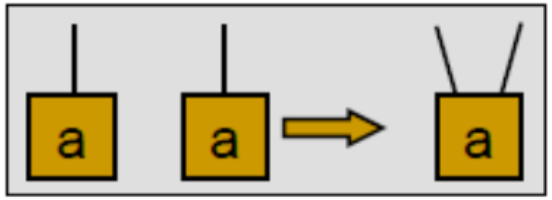
$$f(x_1, x_2, x_3) = \sim x_1 \sim x_2 \sim x_3 + \sim x_1 \sim x_2 x_3 + \sim x_1 x_2 \sim x_3 + x_1 x_2 \sim x_3$$

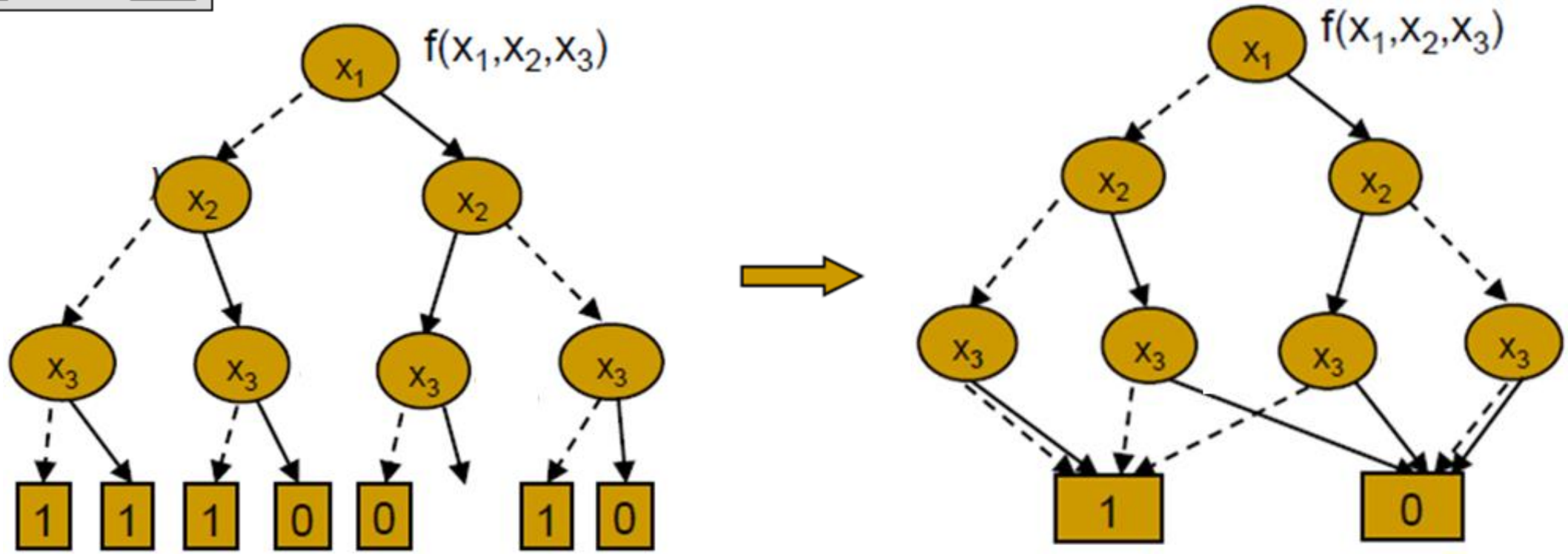
x_1 x_2 x_3	$f(x_1x_2x_3)$
0 0 0	1
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1



Reduced Ordered BDD (ROBDD)

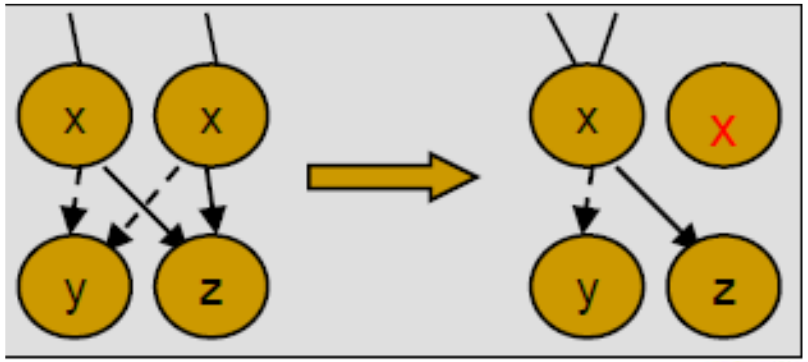
- **BDDs can be very big**
  - Use reduced representation
- **Reduction Rule 1: Merge equivalent leaves**

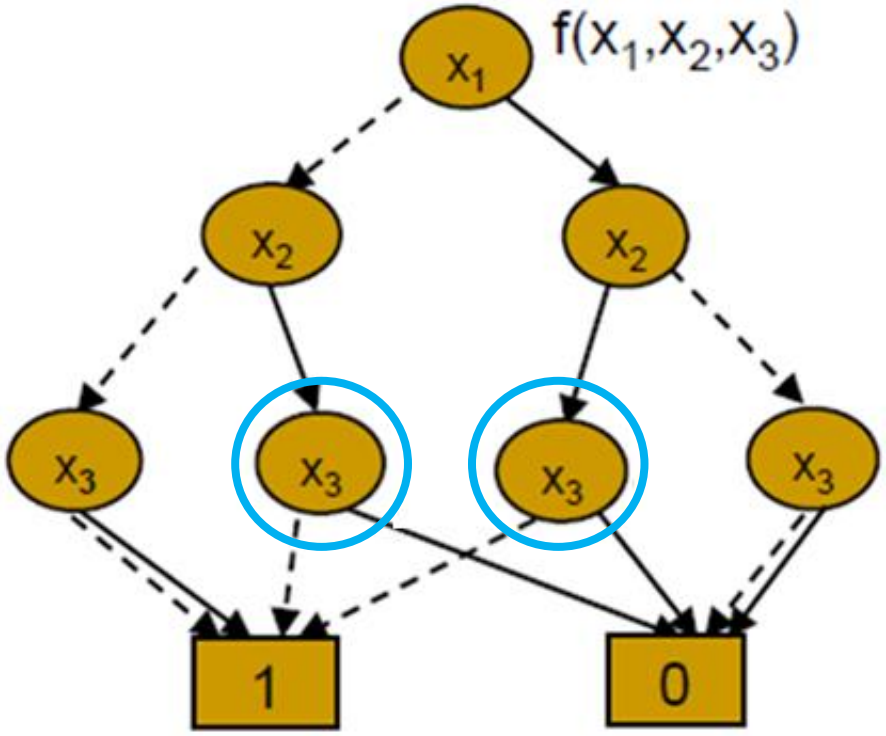


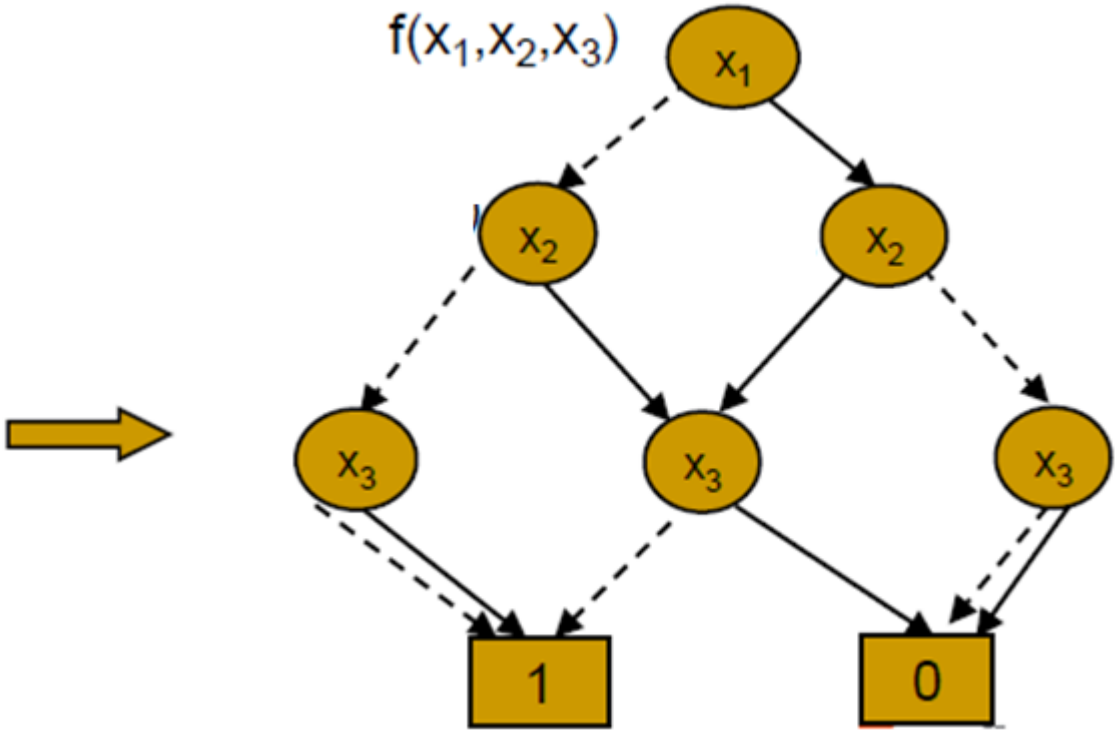


Reduced Ordered BDD (ROBDD)

- **Reduction Rule 2: Merge isomorphic nodes**

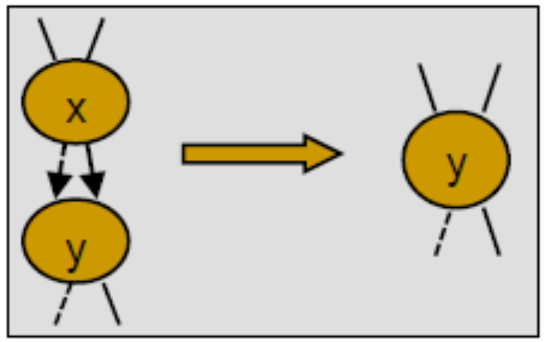


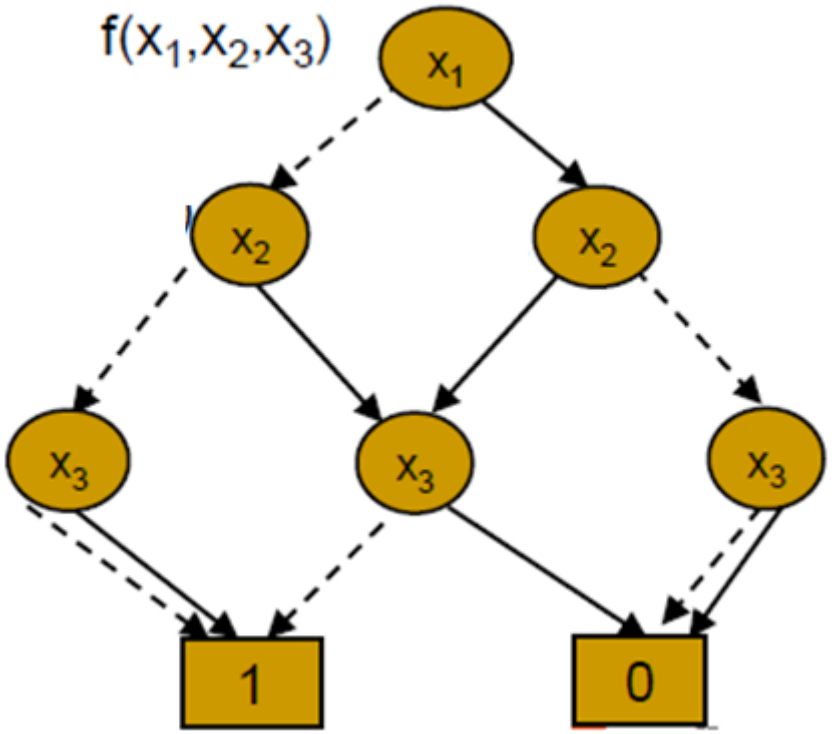


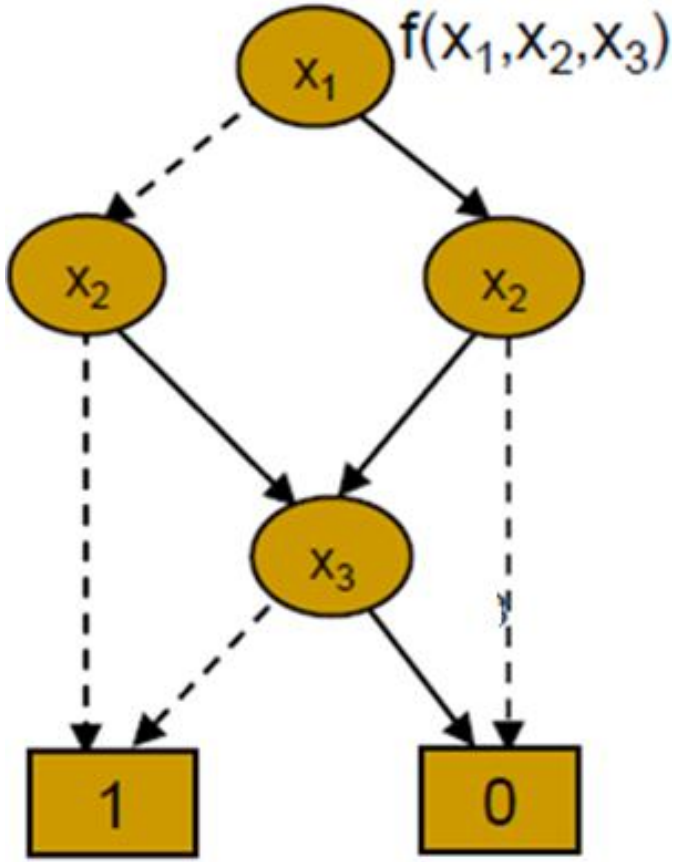


Reduced Ordered BDD (ROBDD)

- **Reduction Rule 3: Eliminate Redundant Tests**





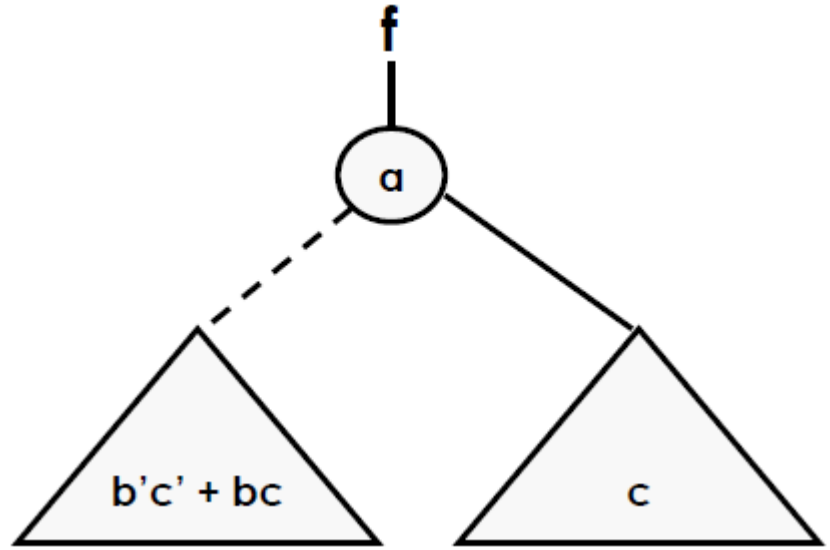


Binary Decision Diagrams (BDD)

- **The Shannon Expansion of a function relates the function to its *cofactors***
  - Given a Boolean function $f(x_1,x_2,\ldots,x_i,\ldots,x_n)$
  - Positive cofactor: $f_{i1} = f(x_1,x_2,\ldots,1,\ldots,x_n)$
  - Negative cofactor: $f_{i0} = f(x_1,x_2,\ldots,0,\ldots,x_n)$
- **Shannon's expansion theory states that**
  - $f = x_i' f_{i0} + x_i f_{i1}$
  - $f = (x_i + f_{i0})(x_i' + f_{i1})$
- **This leads to the formation of a BDD:**
- Example: $f = ac + bc + a'b'c'$

  $= a' (b'c' + bc) + a (c + bc)$

  $= a' (b'c' + bc) + a (c)$



Reduced Ordered BDD (ROBDD)

- Assignment f = ab+a’c+a’bd

- **An Important Point:**

- The size of a BDD can vary drastically if the order in which the variables are expanded is changed.

Technology mapping

- **Technology mapping is the phase of logic synthesis when gates are selected from a technology library to implement the circuit**
- **Why technology mapping?**
- Straight implementation may not be good.
- For example, F=abcdef as a 6-input AND gate causes a long delay
- Gates in the library are pre-designed, they are usually optimized in terms of area, delay, power, etc.
- Fastest gates along the critical path, area-efficient gates (combination) off the critical path.
- **Can apply a minimum cost tree-covering algorithm to solve this problem**

Technology Mapping Algorithm

- **Using a recursive tree-covering algorithm, a logic network to a technology library can be mapped**
- **Process requires three steps:**
  1. Map netlist and tech library to simple gates
     - Describe the netlist with only NAND2 and NOT gates
     - Describe SC library with NAND2 and NOT gates and associate a cost with each gate
  2. Tree-ifying the input netlist
     - Tree covering can only be applied to trees
     - Split tree at all places, where fanout > 1
  3. Minimum Cost Tree matching
     - For each node in your tree, recursively find the minimum cost target pattern at that node

1. Simple Gate Mapping

- **Apply De Morgan laws to your Boolean function to make it a collection of NAND2 and NOT gates**
- Consider

```
t1 = d + e;
t2 = b + h;
t3 = at2 + c;
t4 = t1t3 + fgh;
F = t4';
```

$$t_1 = d + e = \text{NAND}(\overline{d}, \overline{e})$$

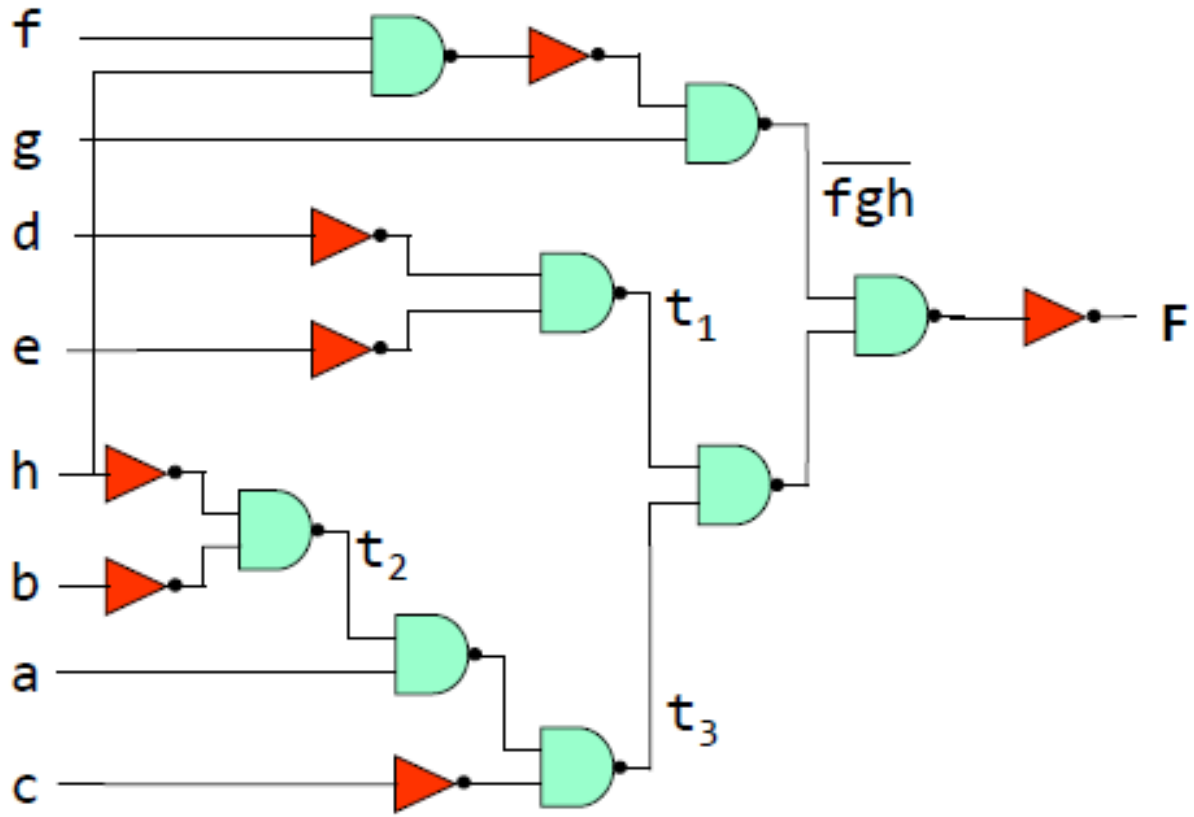
$$t_2 = b + h = \text{NAND}(\overline{b}, \overline{h})$$

$$t_3 = at_2 + c = \overline{\overline{at_2} \cdot \overline{c}} = \text{NAND}(\text{NAND}(a, t_2), \overline{c})$$

$$t_4 = t_1 t_3 + fgh = \text{NAND}(\overline{t_1 t_3}, \overline{fgh})$$

$$fgh = \overline{\overline{fh} \cdot g} = \overline{\overline{\overline{fh} \cdot g}} = \overline{\text{NAND}(\overline{\text{NAND}(f, h)}, g)}$$

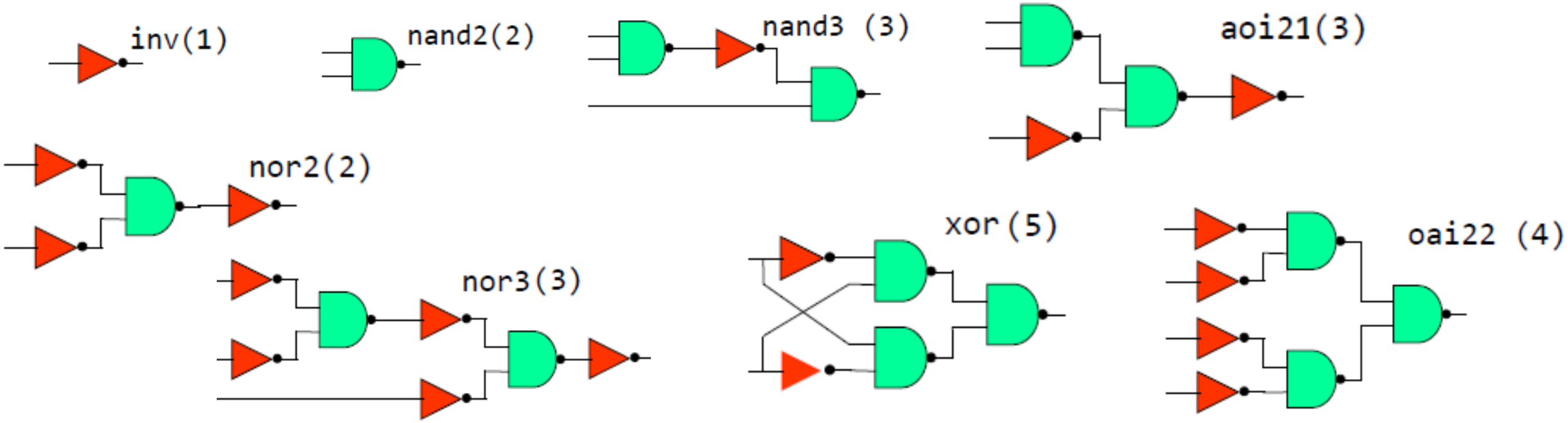
$$F = \overline{t_4}$$



1. Simple Gate Mapping

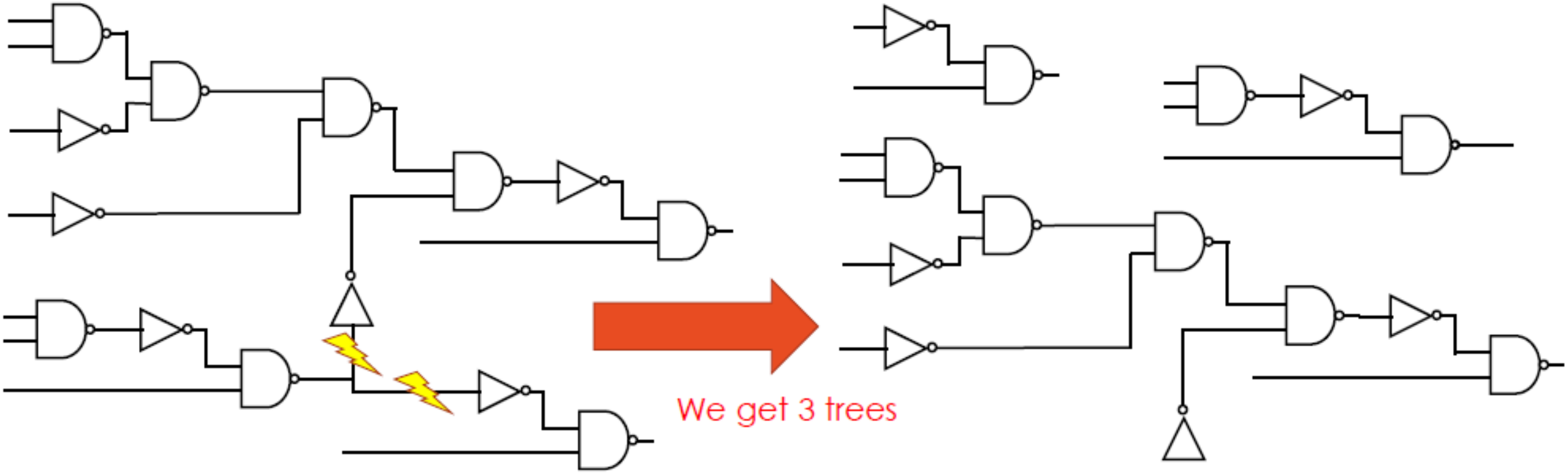
- **Given a set of gates (standard cell library) with cost metrics (area/delay/power)**

- **We need to define the gates with the same NAND2/NOT set**



2. Tree-ifying

- *Break the tree* at any node with fanout>1

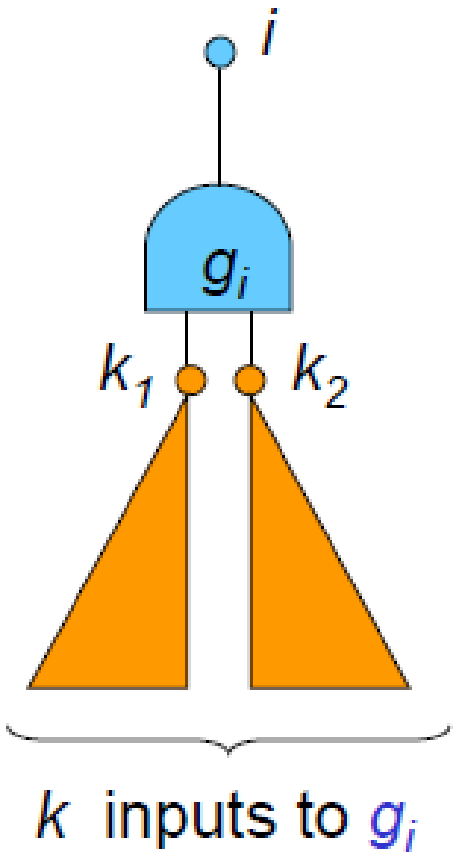


3. Minimum Tree Covering

- **Now, we can apply a recursive algorithm to achieve a minimum cover**
- Start at the output of the graph
- For each node, find all the matching target patterns.
- The cost of node *I* for using gate *g* is:

$$\text{cost}(i) = \min_k \left\{ \text{cost}(g_i) + \sum_k \text{cost}(k_i) \right\}$$

where k_i are the inputs to gate *g*

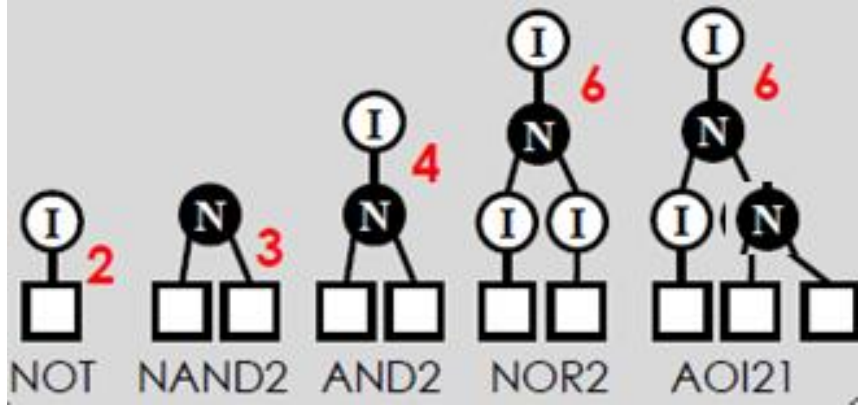


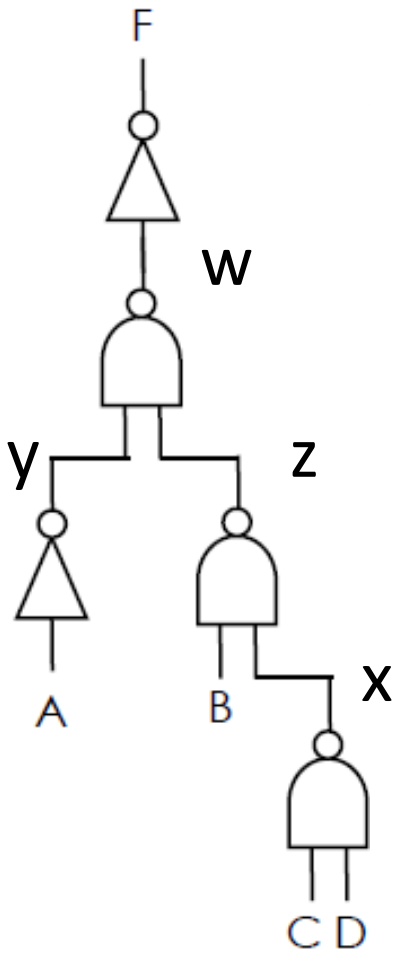
- **For simplicity, redraw the graph with**
- Every NOT is just an empty circle: 

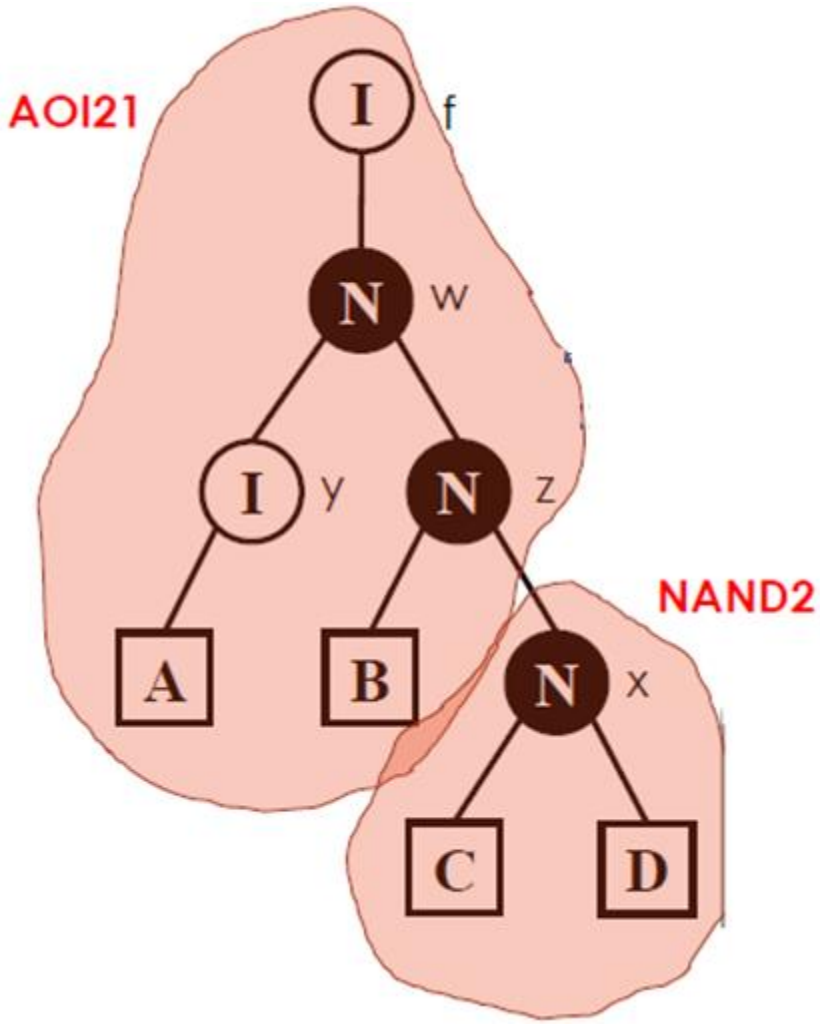
- Every NAND is just a full circle: 

- Every input is just a box: 

Minimum Tree Covering - Example







f: NOT 2 + min(w) = 2 + 11 = 13

AND2 4 + min(y)+min(z) = 4 + 2 + 6 = 12

AOI21 6 + min(x) = 6 + 3 = 9

w: NAND2 3 + min(y)+min(z) = 3 + 2 + 6 = 11

y: NOT 2

z: NAND2 3 + min(x) = 3 + 3 = 6

x: NAND2 3

Assignment

$t_1 = d + e;$
$t_2 = b + h;$
$t_3 = at_2 + c;$
$t_4 = t_1t_3 + fgh;$
$F = t_4';$

$t_1 = a + bc;$
$t_2 = d + e;$
$t_3 = ab + d;$
$t_4 = t_1t_2 + fg;$
$t_5 = t_4h + t_2t_3;$
$F = t_5';$