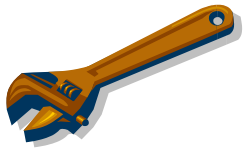
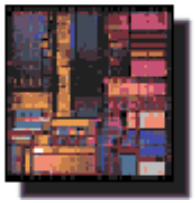






ECE 595Z
Digital VLSI Design Automation

Module 5 (Lectures 14-20): Multi-level Synthesis
Lecture 18



Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

Lecture 17: Re-cap

- **Can we go beyond the Algebraic Model?**

- The algebraic model enables efficient (**fast**) transformations of a Boolean network
  - Collapsing/elimination, extraction/decomposition, substitution, …
- However, it is limited in the **scope of optimization** since it does not take advantage of the unique properties of Boolean algebra.

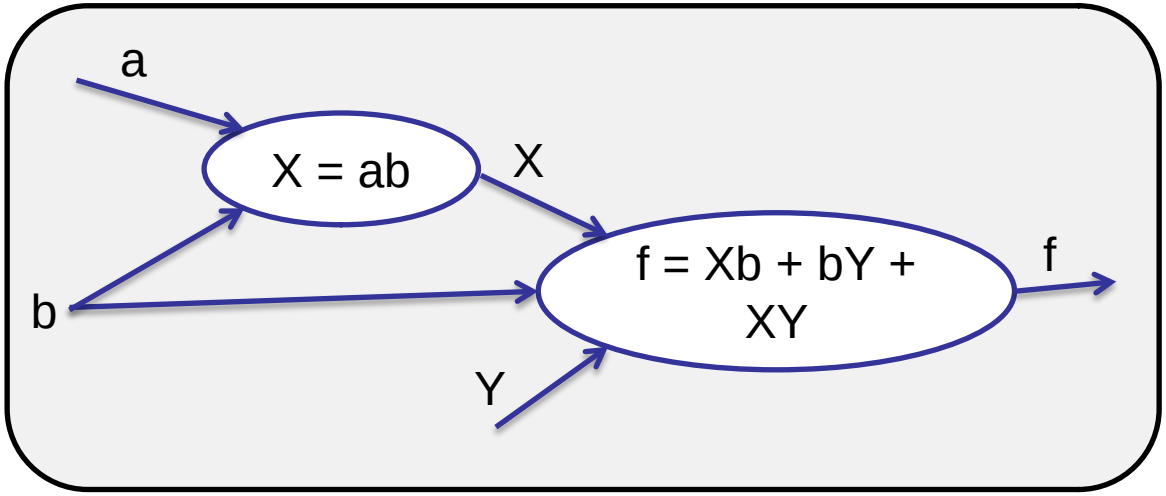
Lecture #17: Re-cap

- Boolean Division
  - Boolean division and factoring are much harder than algebraic counterparts
  - Number of Boolean divisors/factors can be very large
  - Boolean division can be indirectly performed using 2-level minimization
- Boolean Optimization Using Implicit Don't Cares
  - Implicit don't cares are introduced due to the network itself
  - We saw how to derive don't cares at a node's inputs by looking at it's fanin nodes
  - A node can be simplified using its implicit don't cares

Informal Introduction to DCs: Example

- What if we know a little about the network structure?
  - Look at "impossible" values of a, b, X
  - Project to the inputs of the node f
  - Use projected DCs to simplify node f

abX	possible?
000	
001	
010	
011	
100	
101	
110	
111	



XbY	possible?
000	
001	
010	
011	
100	
101	
110	
111	

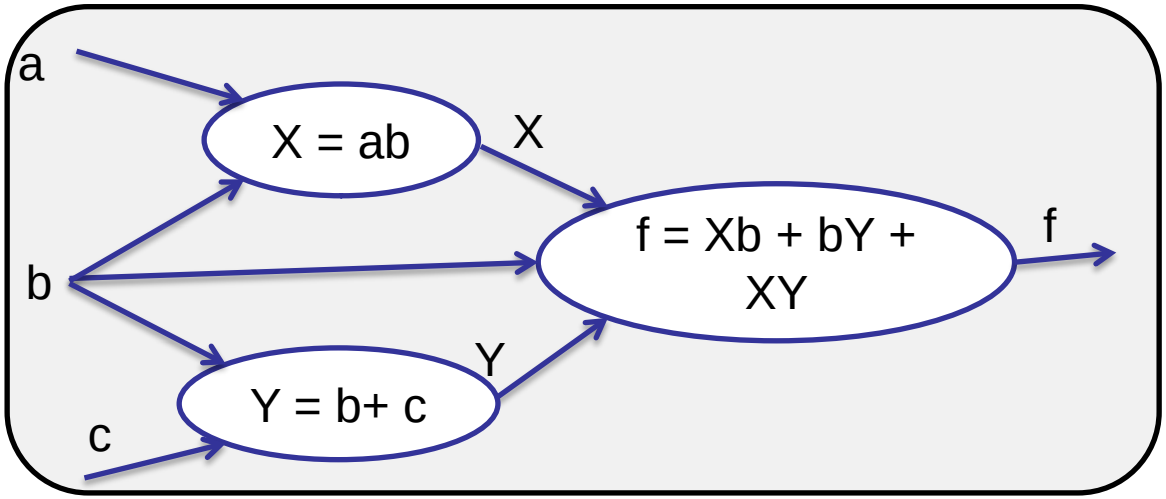
Informal Introduction to DCs

- What if we know even more about the network?

abX	possible?
000	
001	
010	
011	
100	
101	
110	
111	

bcY	possible?
000	
001	
010	
011	
100	
101	
110	
111	

XbY	possible?
000	
001	
010	
011	
100	
101	
110	
111	



Informal Introduction to DCs

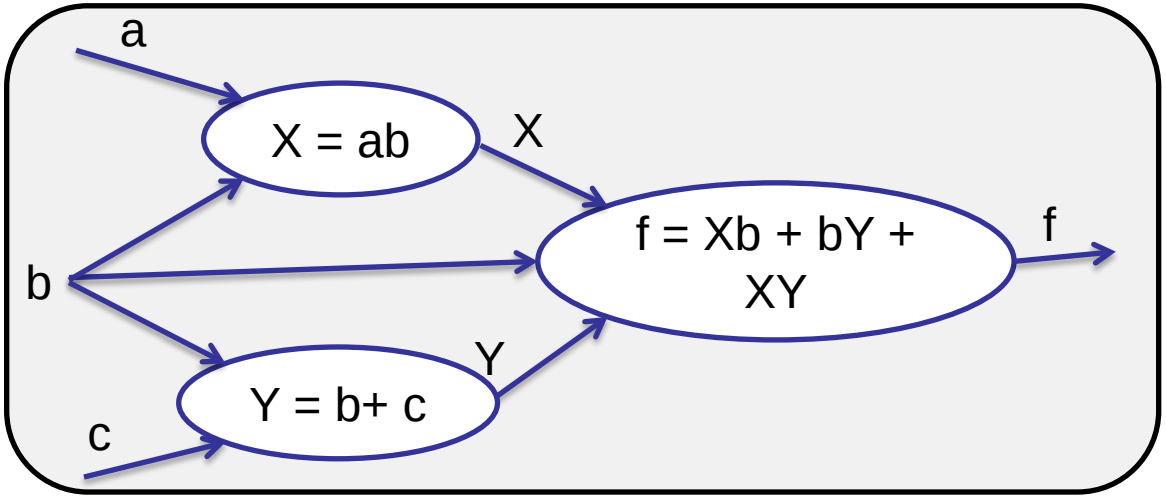
- Can we simplify f further now?

f = Xb + bY + XY

Y \ Xb	00	01	11	10
0			1	
1		1	1	1

Add don't cares & simplify

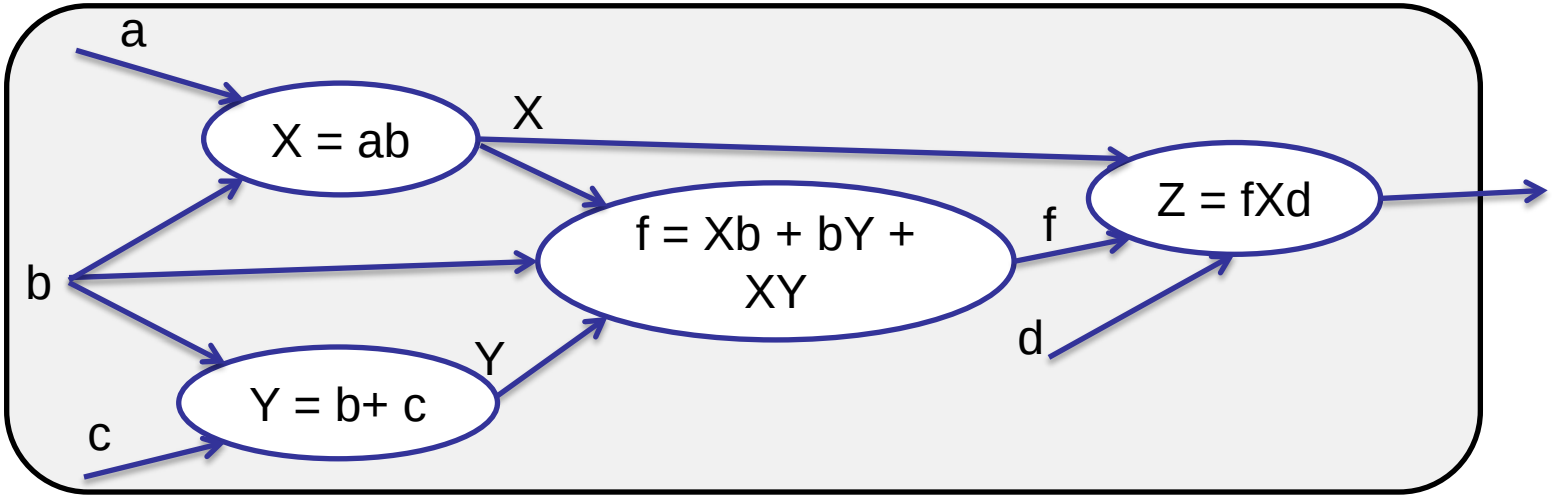
f =



Informal Introduction to DCs

- Let's add information about the nodes "after" f
- When does the output of f actually "matter" to the primary output Z?

	f X d	Z does it change?
Δf	0 0 0	
	1 0 0	
Δf	0 0 1	
	1 0 1	
Δf	0 1 0	
	1 1 0	
Δf	0 1 1	
	1 1 1	

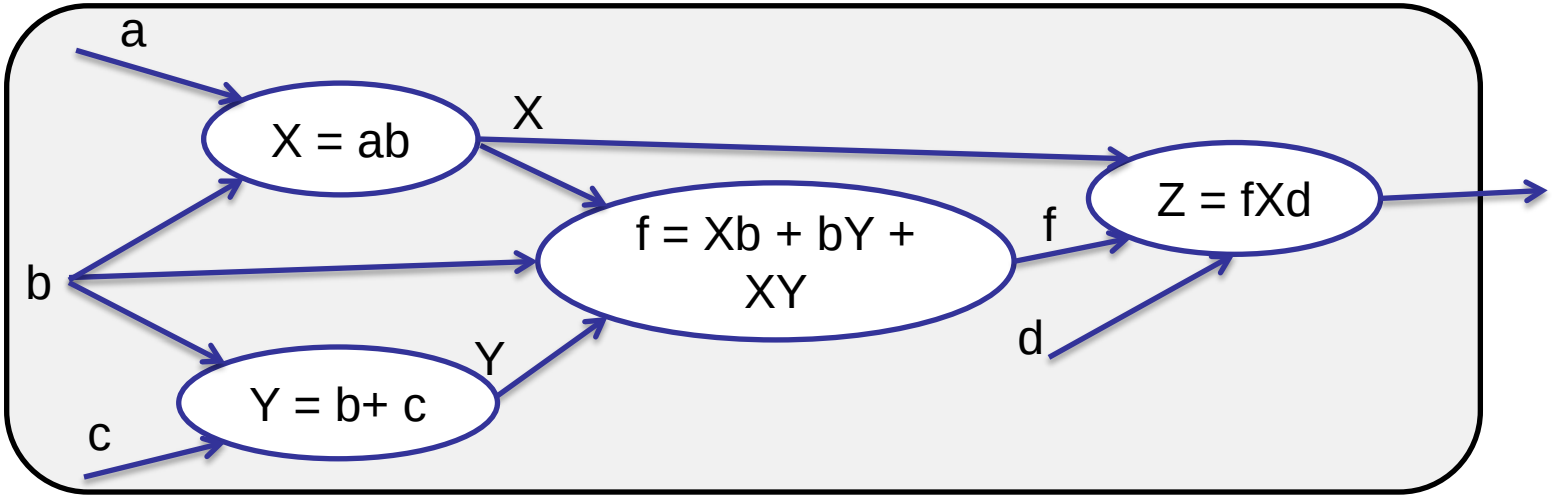


Informal Introduction to DCs

- Which input combinations to f make the output Z insensitive to it?

	f X d	Z does it change?
Δf	0 0 0	
	1 0 0	
Δf	0 0 1	
	1 0 1	
Δf	0 1 0	
	1 1 0	
Δf	0 1 1	
	1 1 1	

XbY	Z insensitive to f?
000	
001	
010	
011	
100	
101	
110	
111	



Informal Introduction to DCs

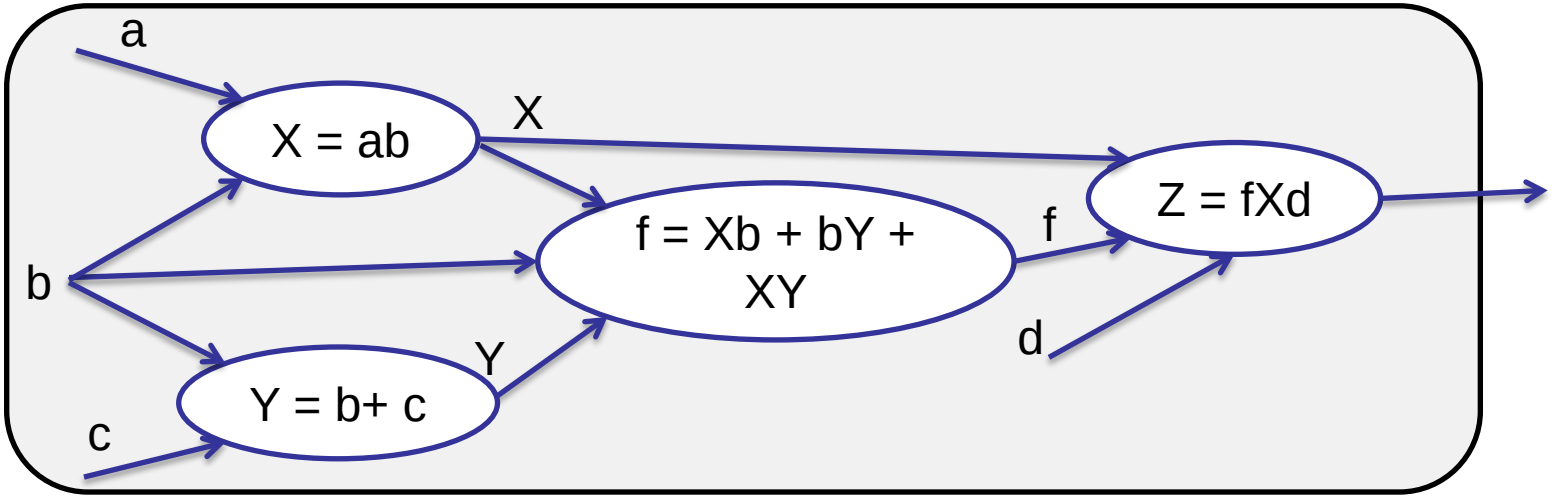
- Can we simplify f further now using ALL the don't cares we have identified?

f = Xb + bY + XY

Y \ Xb	00	01	11	10
0			1	
1		1	1	1

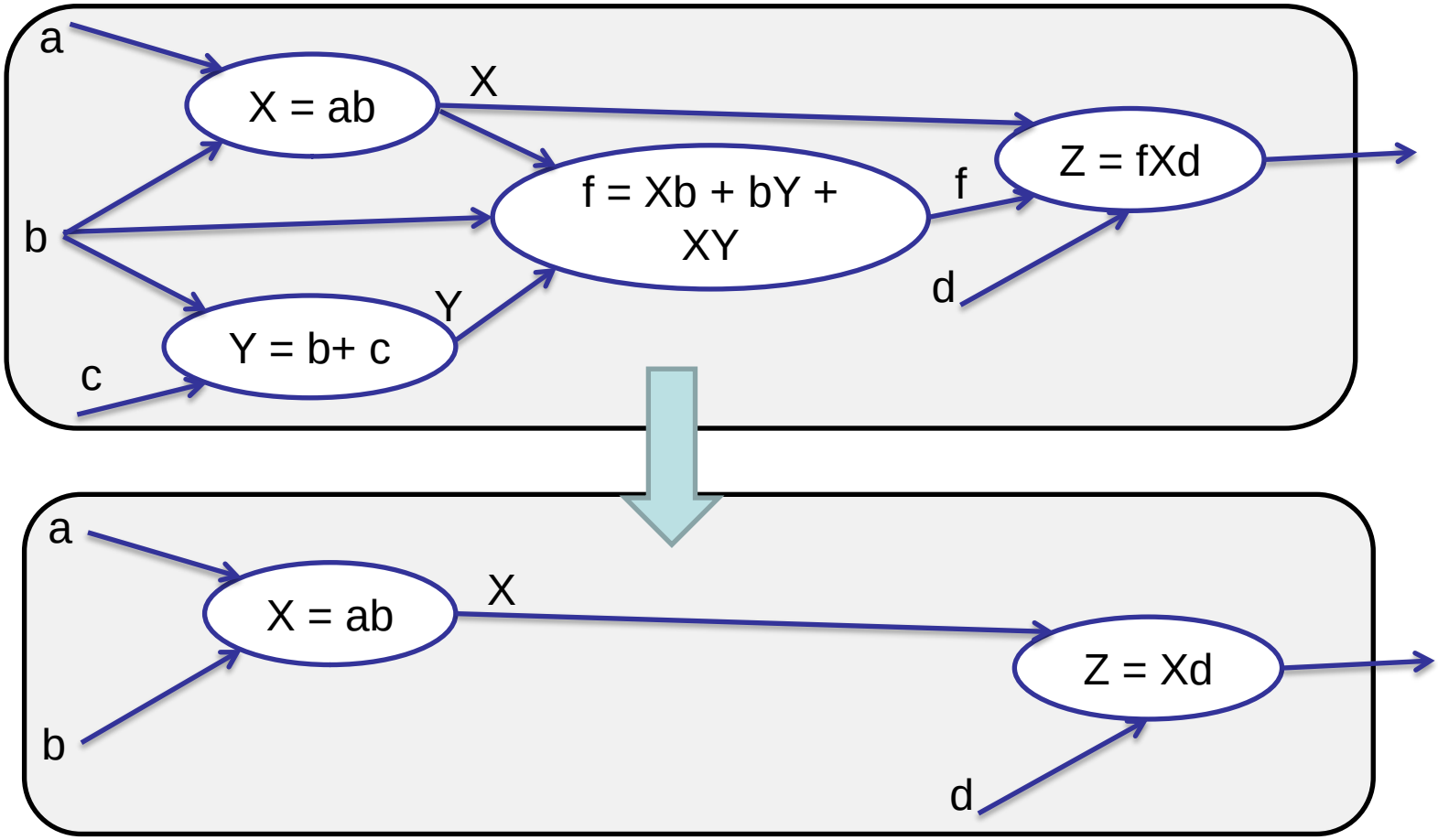
Add don't cares & simplify

f =



Informal Introduction to DCs

- What happened?
  - f and Y were redundant due to the context imposed by the Boolean network!

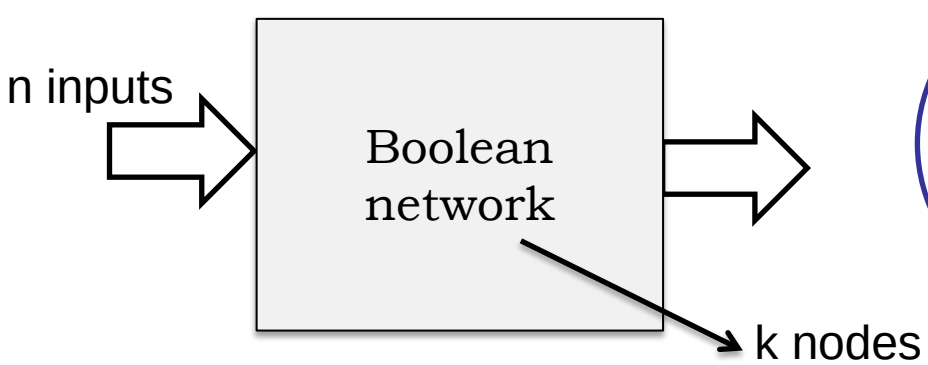


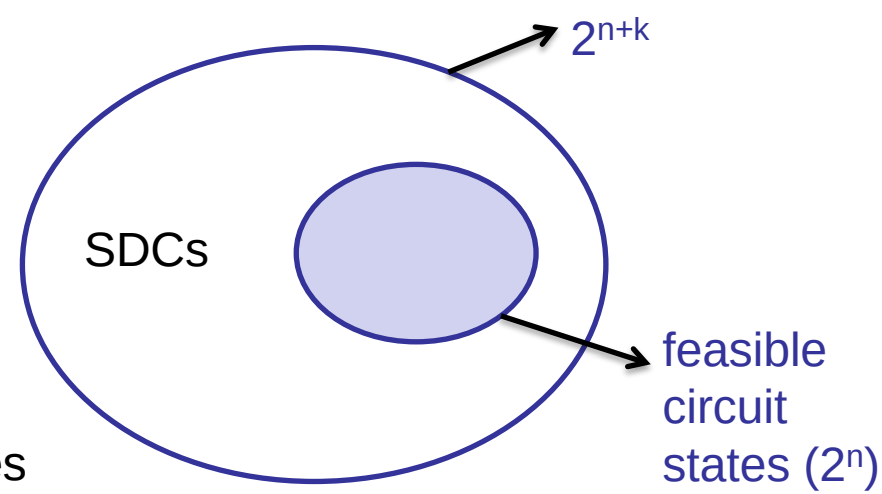
Summary : Implicit Don't Cares

- Arise due to constraints imposed by the Boolean network on each of it's nodes
- Satisfiability don't cares
  - Caused due to combinations of input values to a node that can never occur
- Observability don't cares
  - Caused due to input combinations that make the node's output un-observable at the circuit's primary outputs

Satisfiability Don't Cares

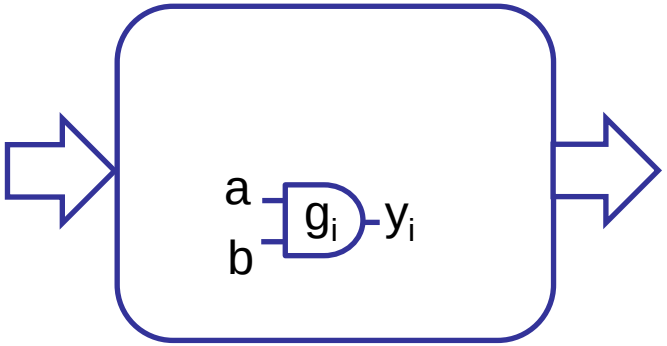
- Given a Boolean network with n primary inputs and k nodes
- The "state" of the circuit is the set of values on all inputs and node outputs
  - $v \in B^{n+k}$
  - However, not all points in B^{n+k} are feasible
    - Only B^n are feasible (input values fully determine circuit state)
  - The remaining points are called **Satisfiability Don't Cares (SDCs)**





Satisfiability Don't Cares

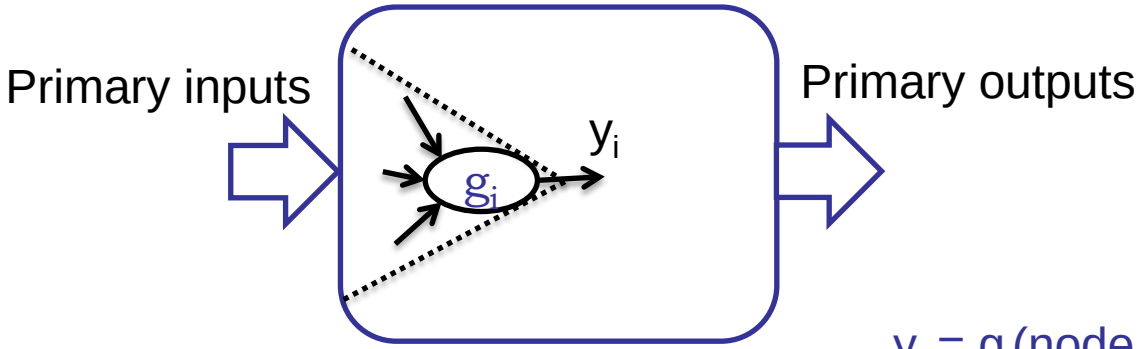
- Finding the SDCs of a Boolean network
  - For each node, list inconsistent assignments of inputs and outputs
    - Example: $y_i = ab$
    - $SDC_i = y_i'ab + y_ia' + y_ib'$
  - $SDC = \cup_i SDC_i$ for $i \in$ all nodes in the network
  - Not specific to a node



- The set of all SDCs is <u>huge</u> for practical sized networks

Observability Don't Cares

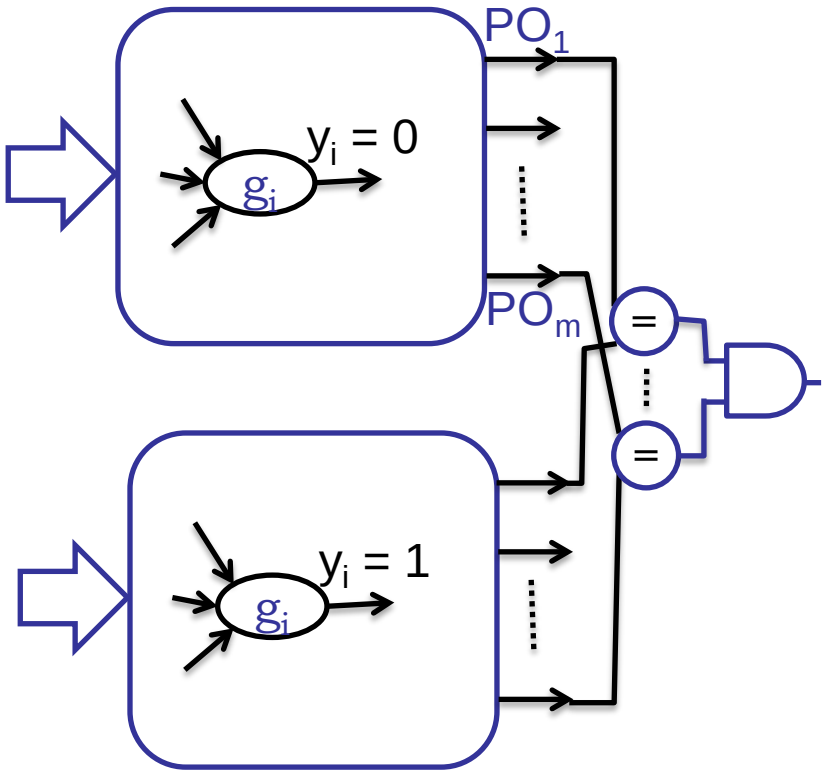
- Some terminology
- Given a node i in a Boolean network
  - Let y_i be a literal representing the output of node i
  - Let g_i be the **local function** at node i, *i.e.*, the node output expressed in terms of the node inputs
  - Let G_i be the **global function** at node i, *i.e.*, the node output in terms of the circuit's primary inputs



$y_i = g_i$(node input vars)
$= G_i$(primary input vars)

Observability Don't Cares

- Remember Co-factors?
  - Here's a chance to apply them!
- Let PO_j, $j \in \{1 \ldots m\}$ be the primary outputs
- For input values where $(G_{PO_j})_{y_i} = (G_{PO_j})_{y_i'}$, the node output is not observable at PO_j
  - This is an **observability don't care** for node i w.r.t PO_j
- The ODC for node i is given by
  - $ODC_i = \cap_j ((G_{PO_j})_{y_i} = (G_{PO_j})_{y_i'})$

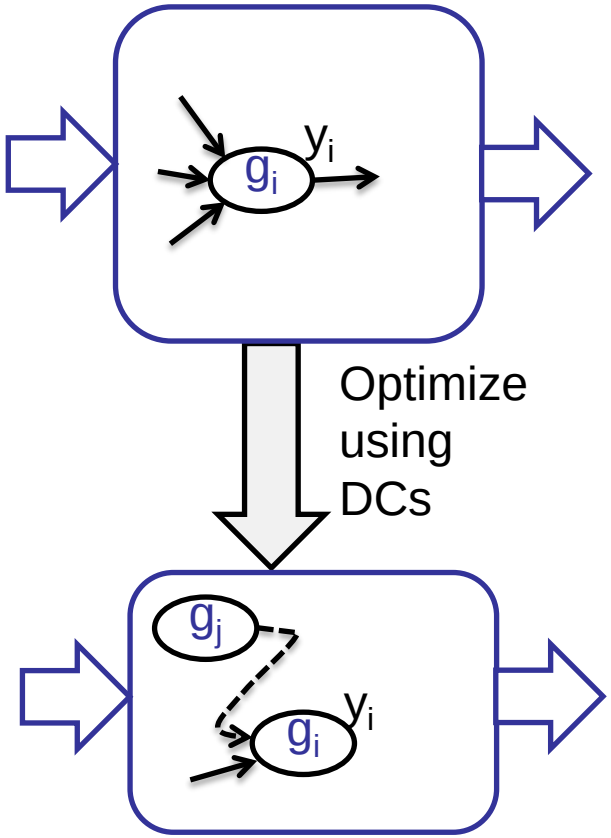


Optimization Using Implicit Don't Cares

- General idea: Use two-level minimization for the SOP expressions in each node of the network
  - Boolean division
  - Node simplification

Boolean Division using DCs

- For a given node in a Boolean network
  - g_i: local function at the node
  - SDC and ODC_i (in terms of all the variables of the network)
- Think of all nodes as functions of all the variables (not just the local inputs)
- Perform two-level minimization using SDC ∪ ODC_i as the don't care set
- Net effect of this is simultaneous Boolean division of g_i by all other nodes in the network
- **PROBLEM**: This can introduce combinational cycles into the network!
  - **Solution**: When optimizing node i, do not include SDC_i and SDCs of all the nodes in the transitive fanout of node i



Trivial result:
$y_i = y_i$

Node Simplification using DCs

- Again, use two-level minimization using SDC and ODC
  - To perform (local) node optimization, DCs need to be specified only in terms of the node's inputs
  - However, SDC and ODC_i are in terms of ALL variables in the network!
    - Need to eliminate variables other than node inputs

- **Question**: Which operations (that we learned in class) have the effect of eliminating a variable from a function?
  - Answers: _____
- **Question**: Which one should we use here?
  - Answer: _____

Quantification

- Two more functions of Shannon co-factors
  - $f_{x_i} \cdot f_{x_i'} = 1$ specifies when f = 1 independent of the value of x_i

    $f(x_1 \ldots x_{i-1}, \mathbf{x_i = 1}, x_{i+1} \ldots x_n) = 1$ AND

    $f(x_1 \ldots x_{i-1}, \mathbf{x_i = 0}, x_{i+1} \ldots x_n) = 1$

  - Called **Universal quantification** or **Consensus**

$$\forall x(f) = f_x \cdot f_{x'}$$

$C_a(f)$

  - $f_x + f_{x'} = 1$ specifies when f = 1 for at least one value of x_i

    $f(x_1 \ldots x_{i-1}, \mathbf{x_i = 1}, x_{i+1} \ldots x_n) = 1$ OR

    $f(x_1 \ldots x_{i-1}, \mathbf{x_i = 0}, x_{i+1} \ldots x_n) = 1$

  - Called **Existential quantification** or **Smoothing**

$$\exists x(f) = f_x + f_{x'}$$

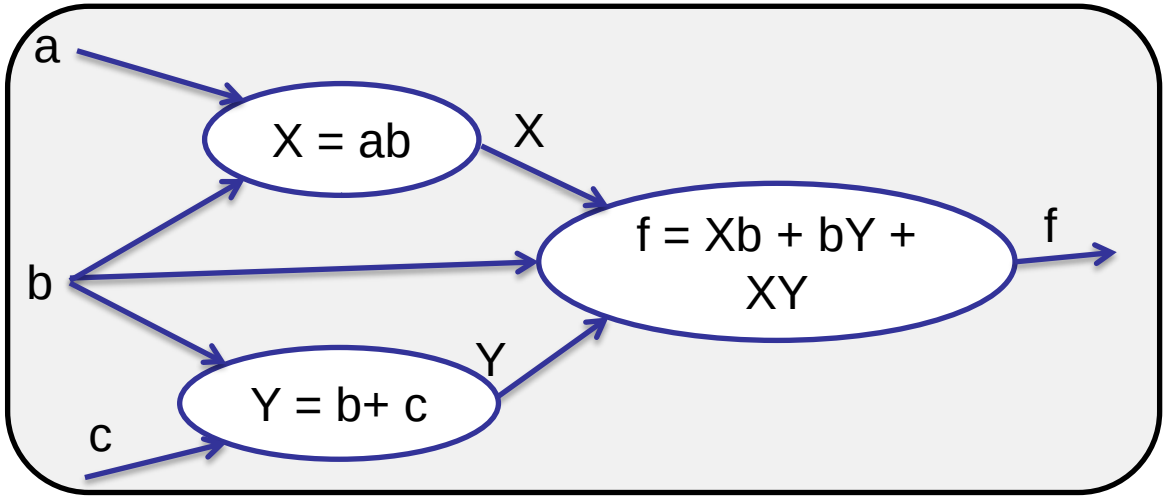
$S_a(f)$

Node Simplification using DCs: Example

- Example
  - $SDC_X = Xa' + Xb' + X'ab$
    - Eliminating a from SDC_X :
  - $SDC_Y = Yb'c' + Y'b + Y'c$
    - Eliminating c from SDC_Y :

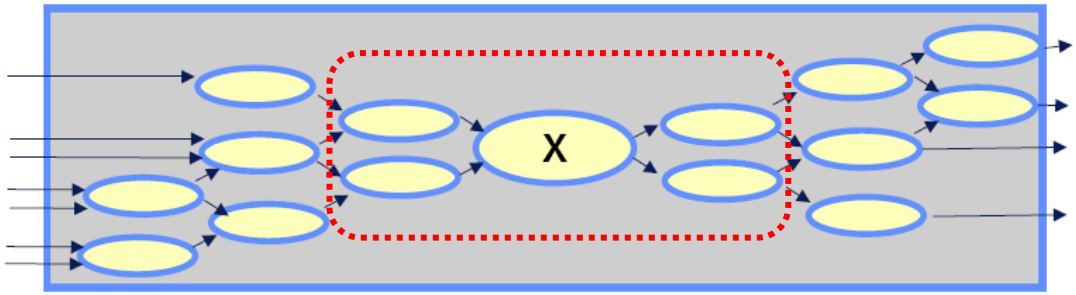
DCs used to simplify node f :

_____



Practical Considerations

- Computing and representing the entire set of SDCs and ODCs can be very slow and require a lot of memory
  - Even using implicit representations like BDDs
- In practice
  - Do not attempt to compute the complete DC set.
    - Look at a limited size "environment" of the node
  - Use "filters" to only compute relevant DCs.



Further Reading : Don't Cares

- De Micheli, Chapter 8.4.1
- Hachtel & Somenzi, Chapter 11.3

- NOTE: De Micheli uses the term "Controllability Don't Cares" for SDCs
  - Strictly speaking, CDCs = SDCs $\cup$ Explicit don't cares specified on primary inputs

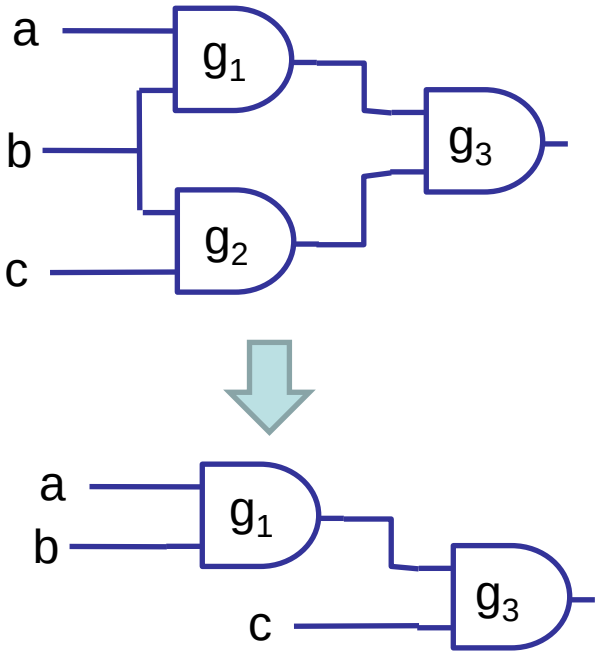
Prime and Irredundant Networks

- Recall that each node in the Boolean network is represented by an SOP expression or a cover
- The node is **locally prime and irredundant (PI)** if no literal or cube can be deleted from the SOP expression without changing the local function
- The node is **prime and irredundant w.r.t. the entire network** if no literal or cube can be deleted from the SOP expression without changing the *global function for some primary output*
- Question: Which of the above two conditions is stricter?

- A **Boolean network is prime and irredundant** if each node is PI w.r.t. the network

Prime and Irredundant Networks

- Example



Are g1, g2, and g3 locally prime and irredundant?

Are g1, g2, and g3 prime and irredundant w.r.t. the network?

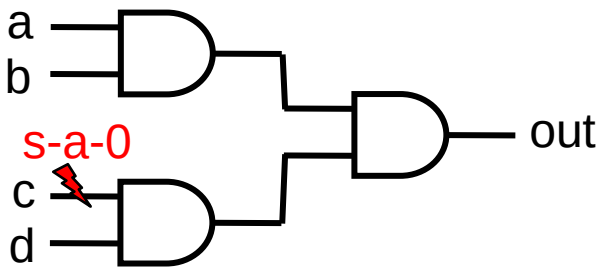
Prime and Irredundant Networks

- **Theorem**: If the SOP expression for a node g_i in a Boolean network is made prime and irredundant using SDC and ODC_i, then it is prime and irredundant w.r.t. the network

- Prime and irredundant networks are desirable for multiple reasons

Multi-level Minimization and Testability

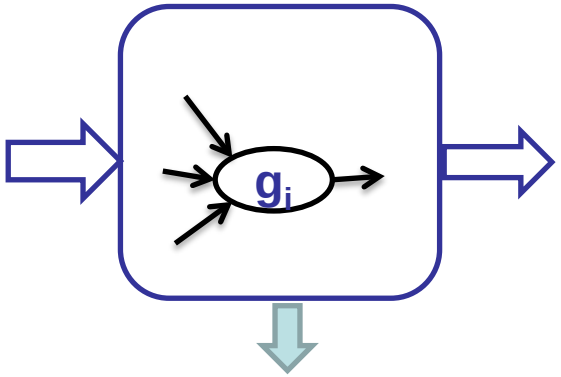
- Recall that we would like to test fabricated instances for manufacturing defects
  - Need to generate test vectors
  - Commonly used: Stuck-at fault model

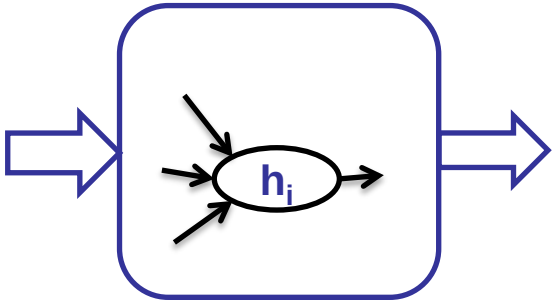


- **Theorem**: A prime and irredundant Boolean network is 100% testable for single stuck-at faults at all node inputs and outputs
  - Test vector exists to detect each stuck-at fault
- **Theorem**: Algebraic transformations preserve single stuck-at fault testability (and the test set!)

Other Approaches to Boolean Optimization

- Perturbation
  - Change the local function at a node from g_i to h_i
    - OK if $g_i \oplus h_i \subseteq (SDC \cup ODC_i)$





- Redundancy Addition and Removal
  - Identify redundant s-at faults
  - Simplify circuit by propagating "constant" values
  - Add redundant connections to create new opportunities
    - Use don't cares to ensure that added connections do not change a function at the primary outputs