VE527

Computer-Aided Design of Integrated Circuits

Multi-Level Logic Synthesis:

Don’t Cares

Outline

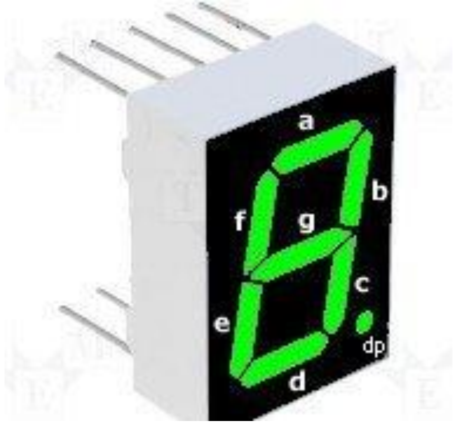
- Implicit Don’t Cares
    - Introduction
    - Method to Obtain them

Don’t Cares

- We made progress on multi-level logic by **simplifying** the model.
  - Algebraic model: we **get rid of** a lot of “difficult” Boolean behaviors.
  - But we lost some optimality in the process.
- How do we put it back? One surprising answer: **Don’t cares**
  - To help this, **extract** don’t cares from “surrounding logic,” use them **inside each node**.
- The big difference in multi-level logic
  - Don’t cares happen as a natural byproduct of Boolean network model: called **Implicit Don’t Cares**.
  - They are all over the place, in fact. Very useful for simplification.
  - But they are **not explicit**. We have to **go hunt for them**…

Don’t Cares Review: 2-Level

- In basic digital design…
  - Don’t Care (DC) = an input pattern that **can never happen** or you don’t care the output if it happens.
  - Example: use **binary-coded decimals (BCD)** to control **seven-segment digital tube**.



x y z w	decimal value	segment a
0 0 0 0	0	1
0 0 0 1	1	0
0 0 1 0	2	1
0 0 1 1	3	1
0 1 0 0	4	0
0 1 0 1	5	1
0 1 1 0	6	1
0 1 1 1	7	1
1 0 0 0	8	1
1 0 0 1	9	1

How about input (x,y,z,w) =(1,0,1,0),(1,0,1,1) …?

Don’t care!

Don’t Cares Review: 2-Level

- Since patterns (x,y,z,w)=(1,0,1,0), (1,0,1,1), (1,1,0,0), (1,1,0,1), (1,1,1,0), (1,1,1,1) are don’t cares, we are **free to decide** whether F=1 or 0, to better **optimize** F.

x y z w	decimal value	segment a
0 0 0 0	0	1
0 0 0 1	1	0
0 0 1 0	2	1
0 0 1 1	3	1
0 1 0 0	4	0
0 1 0 1	5	1
0 1 1 0	6	1
0 1 1 1	7	1
1 0 0 0	8	1
1 0 0 1	9	1

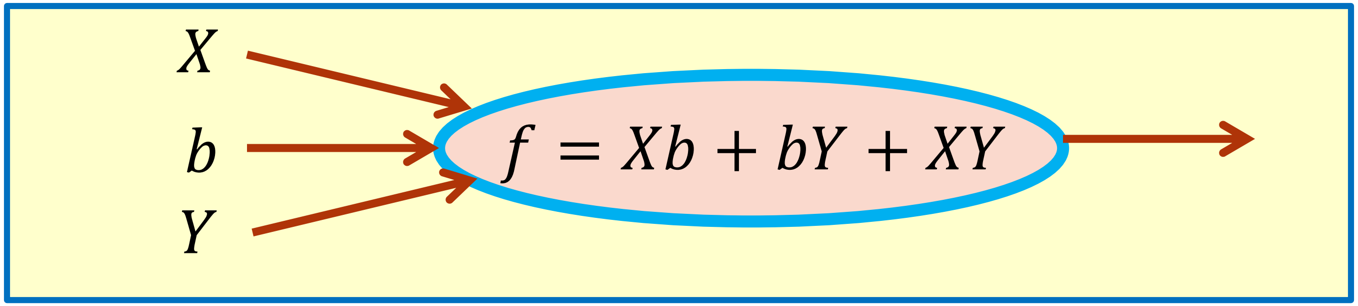
xy zw	00	01	11	10
00	1	0	d	1
01	0	1	d	1
11	1	1	d	d
10	1	1	d	d

Don’t Cares (DCs): Multi-level

- What’s different in multi-level?
  - DCs arise **implicitly**, as a result of the **Boolean logic network structure**.
  - We must go find these implicit don’t cares – we must search for them explicitly.

Multi-level DCs: Informal Tour

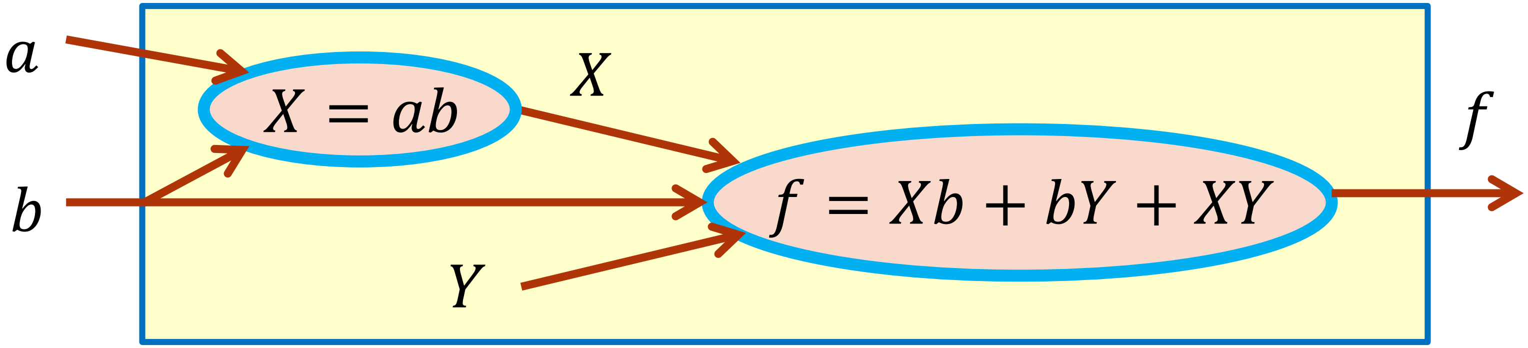
- Suppose we have a Boolean network and a node f in the network.



- Can we say anything about **don't cares** for node f?
  - No. We don't know any "context" for surrounding parts of network.
  - As far as we can tell, all patterns of inputs (X,b,Y) are possible.
  - We **cannot further simplify** the expression for f.

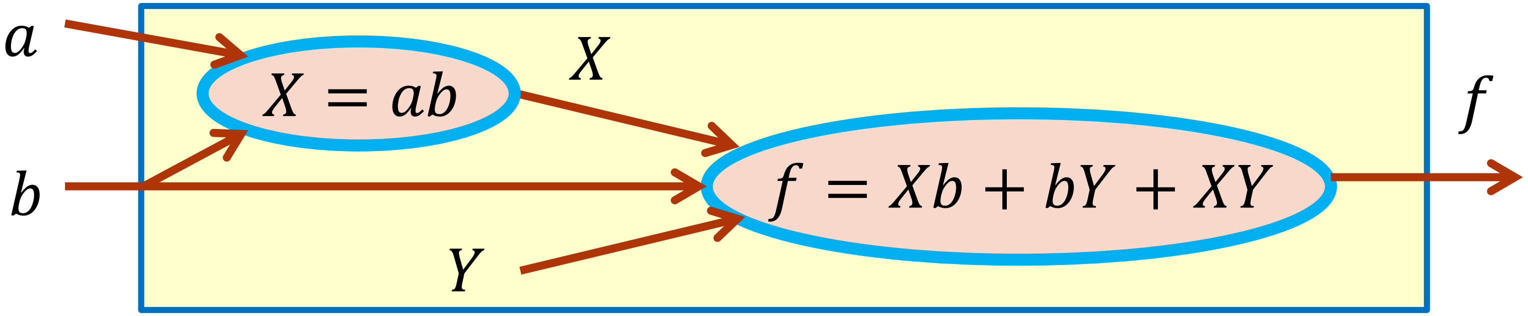
Multi-level DCs: Informal Tour

- Now suppose we know something about input X to f:
  - Node $X = ab$.
  - Also assume a and b are **primary inputs (PIs)** and f is **primary output (PO)**.



- Now can we say something about DCs for node f…?
  - **YES!**
  - Because there are some **impossible patterns** of (X, b, Y).

Multi-level DCs: Informal Tour



a

b

$X = ab$

X

Y

$f = Xb + bY + XY$

f

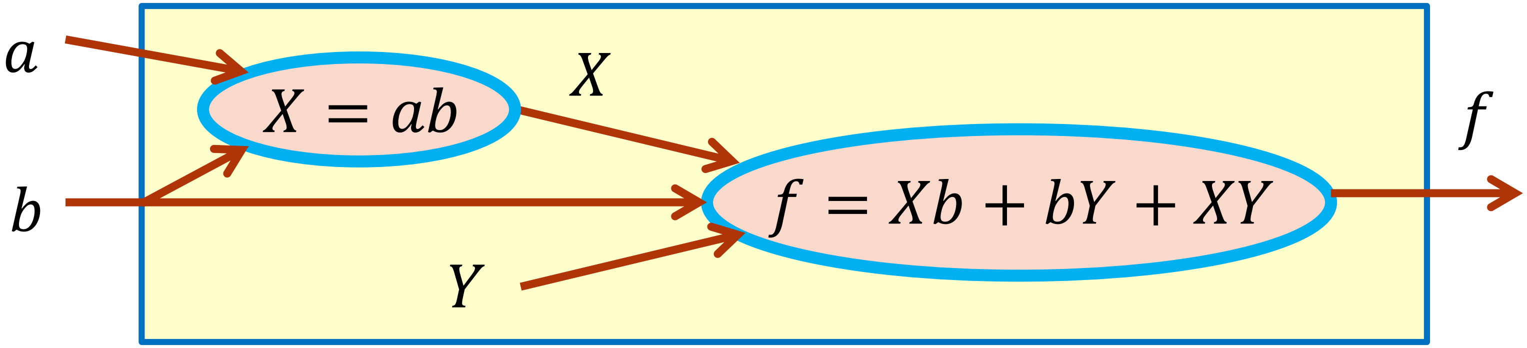
The possible input/output patterns for node X

a	b	X	Can it occur?
0	0	0	Yes
0	0	1	No
0	1	0	Yes
0	1	1	No
1	0	0	Yes
1	0	1	No
1	1	0	No
1	1	1	Yes

b	X	Can it occur?
0	0	Yes
0	1	No
1	0	Yes
1	1	Yes

Impossible patterns for (X, b, Y) are: (1, 0, 0) and (1, 0, 1)

Multi-level DCs: Informal Tour



- Impossible patterns for (X, b, Y) are (1, 0, 0) and (1, 0, 1).
  - With them, we can simplify f.

Can be simplified as
$f = X + bY$

Kmap for $f = Xb + bY + XY$

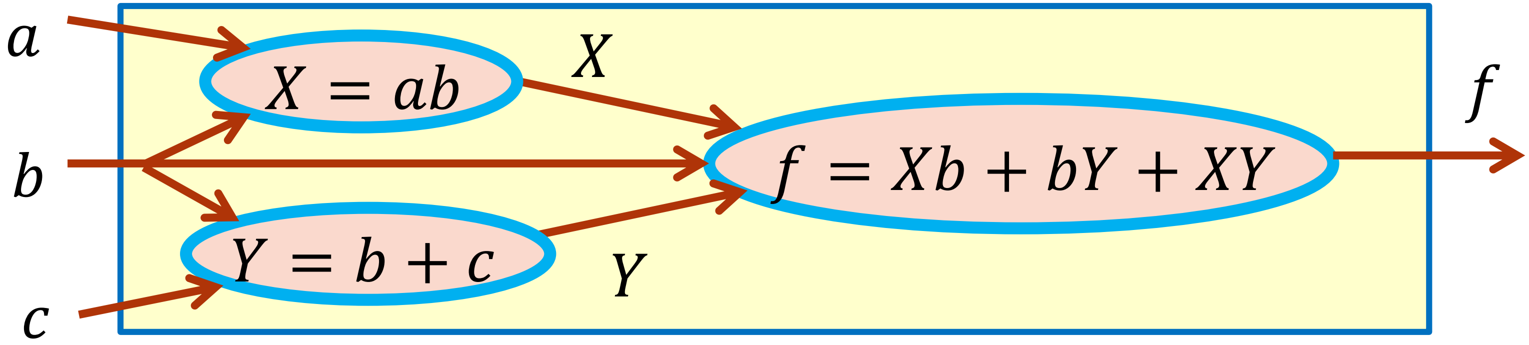
Y \ Xb	00	01	11	10
0			1	
1		1	1	1

With don't cares

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

Multi-level DCs: Informal Tour

- Now further suppose $Y = b + c$. What will happen?

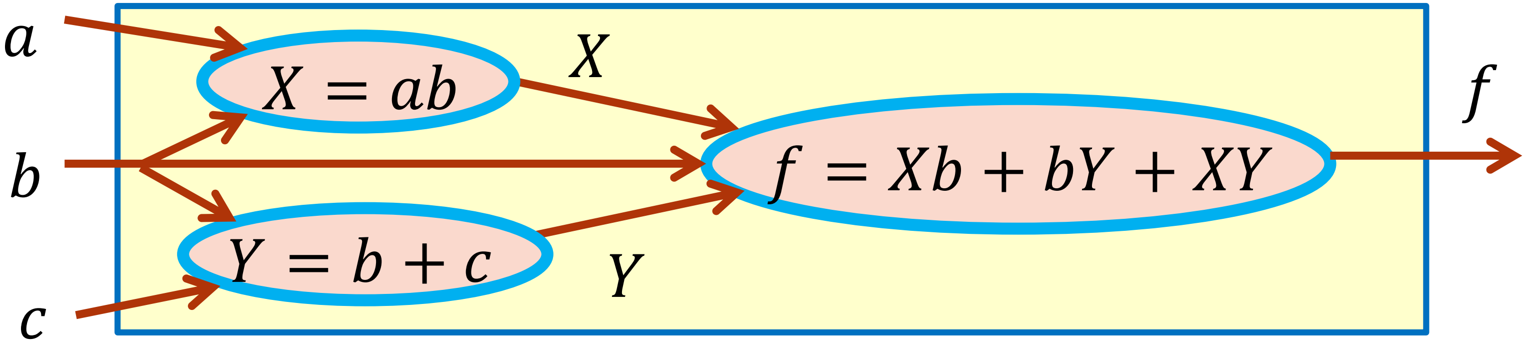


b	c	Y	Can it occur?
0	0	0	Yes
0	0	1	No
0	1	0	No
0	1	1	Yes
1	0	0	No
1	0	1	Yes
1	1	0	No
1	1	1	Yes

b	Y	Can it occur?
0	0	Yes
0	1	Yes
1	0	No
1	1	Yes

Impossible patterns for (X, b, Y) are: (0, 1, 0) and (1, 1, 0)

Multi-level DCs: Informal Tour



- Impossible patterns for (X, b, Y) are
  - (1,0,0), (1,0,1) (From $X = ab$)
  - (0,1,0), (1,1,0) (From $Y = b + c$)

f can be simplified as $f = b$

Kmap for $f = Xb + bY + XY$

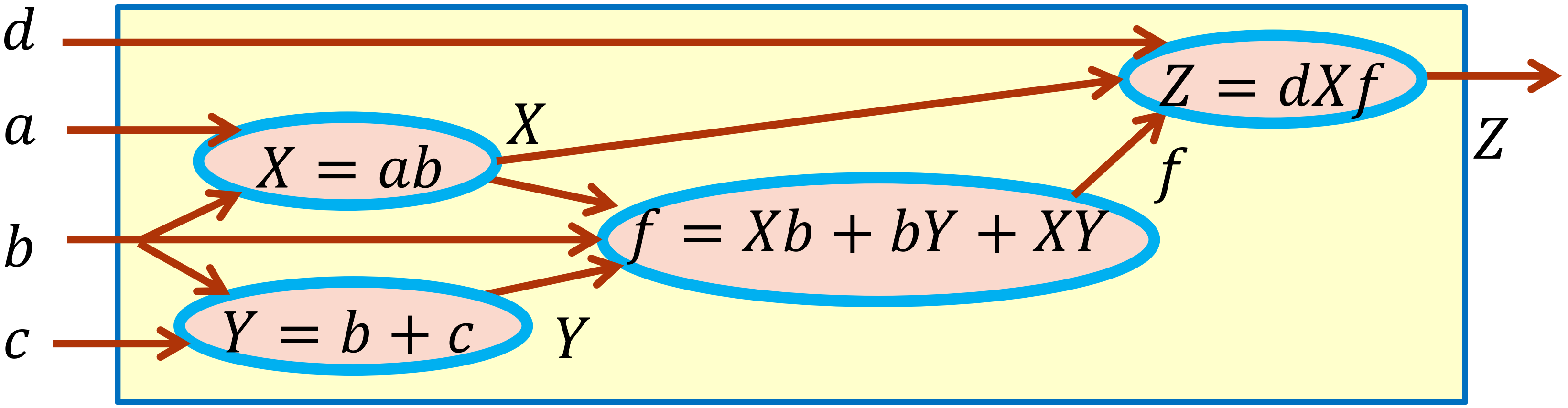
Y \ Xb	00	01	11	10
0			1	
1		1	1	1

With don't cares

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

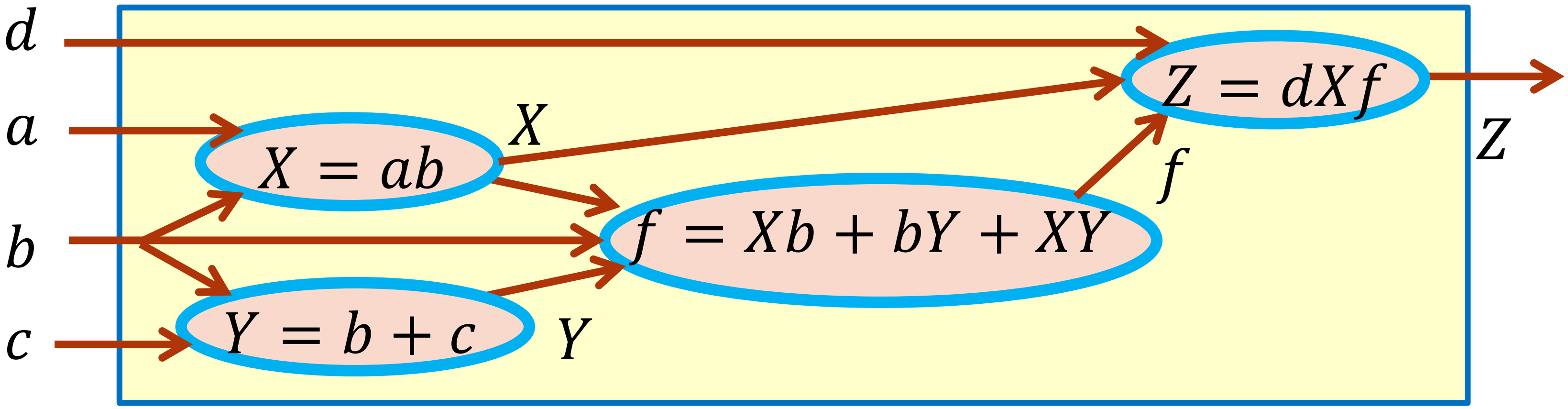
Multi-level DCs: Informal Tour

- Now suppose f is **not** a **primary output**, Z is.



- **Question**: when does the value of the output of node f actually **affect** the primary output Z?
  - Or, said **conversely**: When does it **not matter** what f is?
  - Let's go look at patterns of (f, X, d) at node Z…

When Is Z “Sensitive” to Value of f?



d

a

b

c

$X = ab$

X

$Y = b + c$

Y

$f = Xb + bY + XY$

f

$Z = dXf$

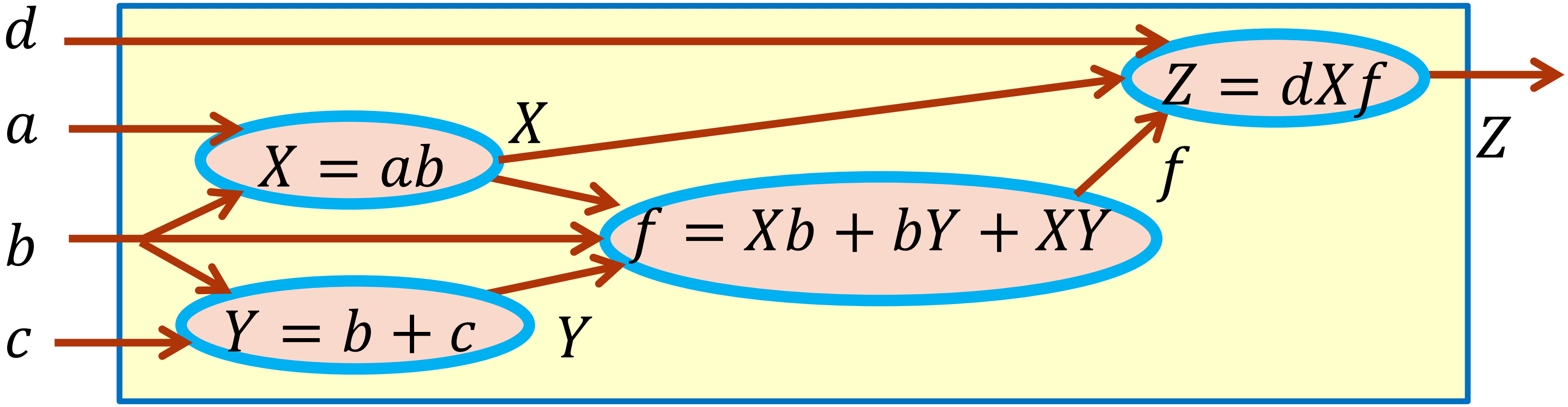
Z

f	X	d	Z	Does f affect Z?
0	0	0	0	No
1	0	0	0	
0	0	1	0	No
1	0	1	0	
0	1	0	0	No
1	1	0	0	
0	1	1	0	Yes
1	1	1	1	

Can we use this information to find new patterns of (X, b, Y) to help us simplify f further?

YES!

When Is Z “Sensitive” to Value of f?



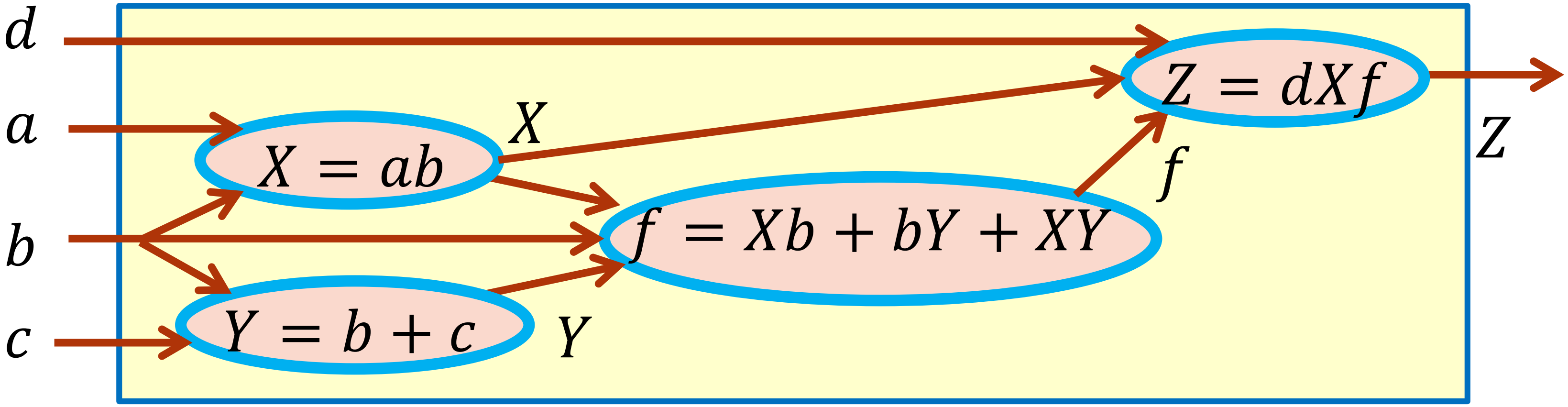
f	X	d	Z	Does f affect Z?
0	0	0	0	No
1	0	0	0	
0	0	1	0	No
1	0	1	0	
0	1	0	0	No
1	1	0	0	
0	1	1	0	Yes
1	1	1	1	

What patterns at **input to** f node (i.e., (X, b, Y)) are DCs, because those patterns make Z output **insensitive** to changes in f?

$$(X, b, Y) = (0, -, -)$$

This means when $X = 0$, we can set f to any value – it **won’t change** Z.
So $(X, b, Y) = (0, -, -)$ is DC of f!

Multi-level DCs: Informal Tour



d

a

b

c

$X = ab$

X

$Y = b + c$

Y

$f = Xb + bY + XY$

f

$Z = dXf$

Z

- So, we can use this **new** DC pattern $(0, -, -)$ to simplify f further…
  - … with previous DC patterns (1,0,0), (1,0,1), (0,1,0), (1,1,0).

Kmap for $f = Xb + bY + XY$

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

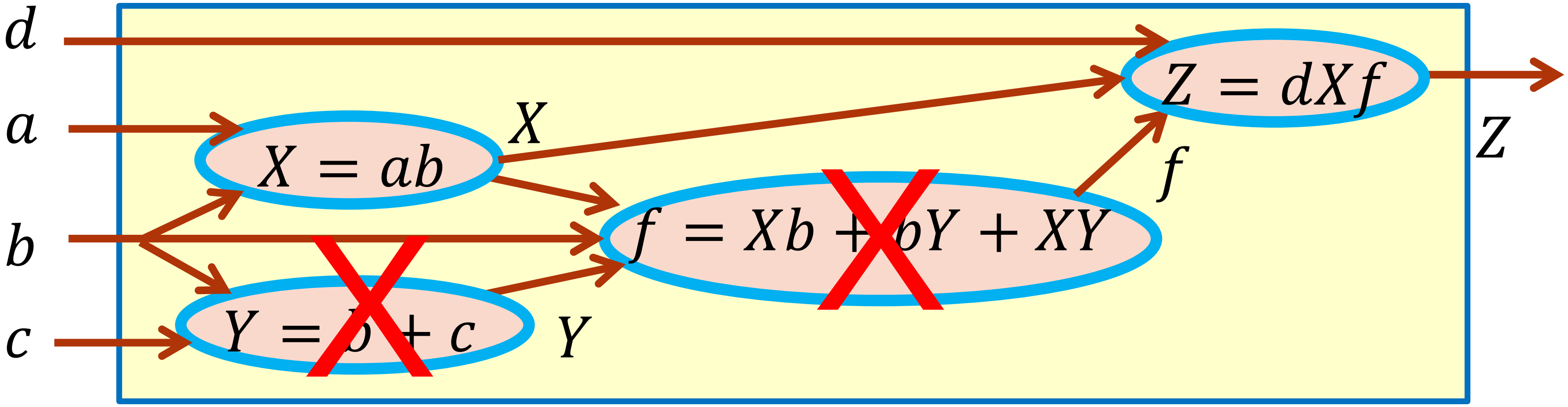
With don’t cares

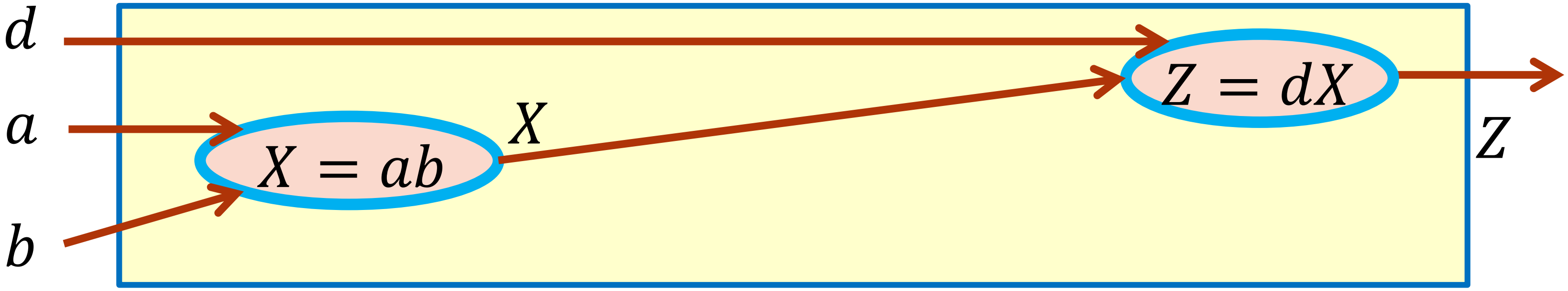
f simplified as 1

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

Final Result: Multi-level DC Tour

- What happened to f?
  - Due to network **context**, it **disappeared** ($f = 1$)!





Summary

- Don’t Cares are **implicit** in the Boolean network model.
  - They arise from the **graph structure** of the multilevel Boolean network model itself.
- Implicit Don’t Cares are **powerful**.
  - They can greatly help simplify the 2-level SOP structure of any node.
- Implicit Don’t Cares require **computational work** to find.
  - For this example, we just “stared at the logic” to find the DC patterns.
  - We need some **algorithms** to do this automatically!
  - This is what we need to study next …

Outline

- Implicit Don’t Cares
  - Introduction
  - Method to Obtain them

3 Types of Implicit DCs

- **Satisfiability** don't cares: **SDCs**
  - Belong to the **wires** inside the Boolean logic network.
  - Used to compute **controllability** don't cares (below).

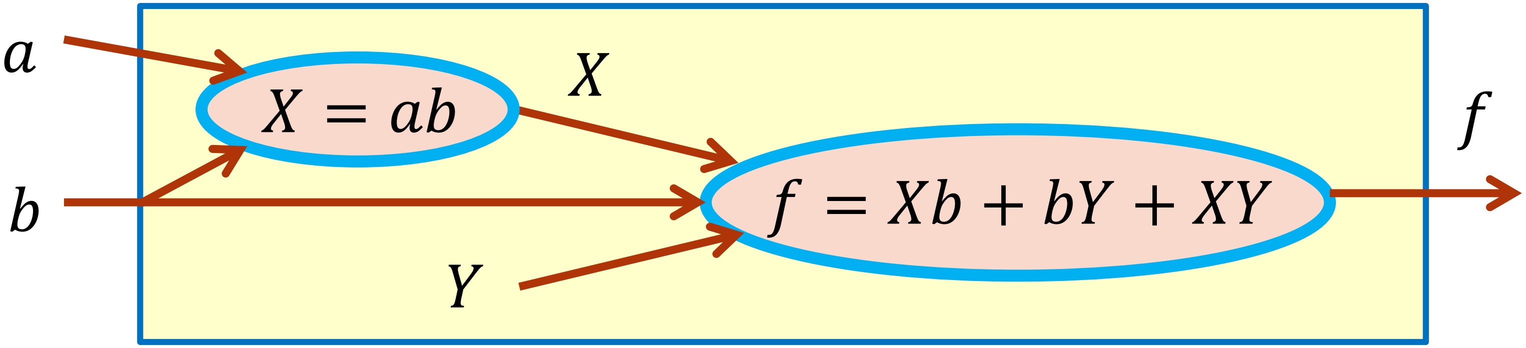
- **Controllability** don't cares: **CDCs**
  - Patterns that **cannot happen at inputs** to a network node.

- **Observability** don't cares: **ODCs**
  - Patterns that "**mask**" outputs.

Controllability don't cares: CDCs

- Patterns that **cannot happen at inputs** to a network node.

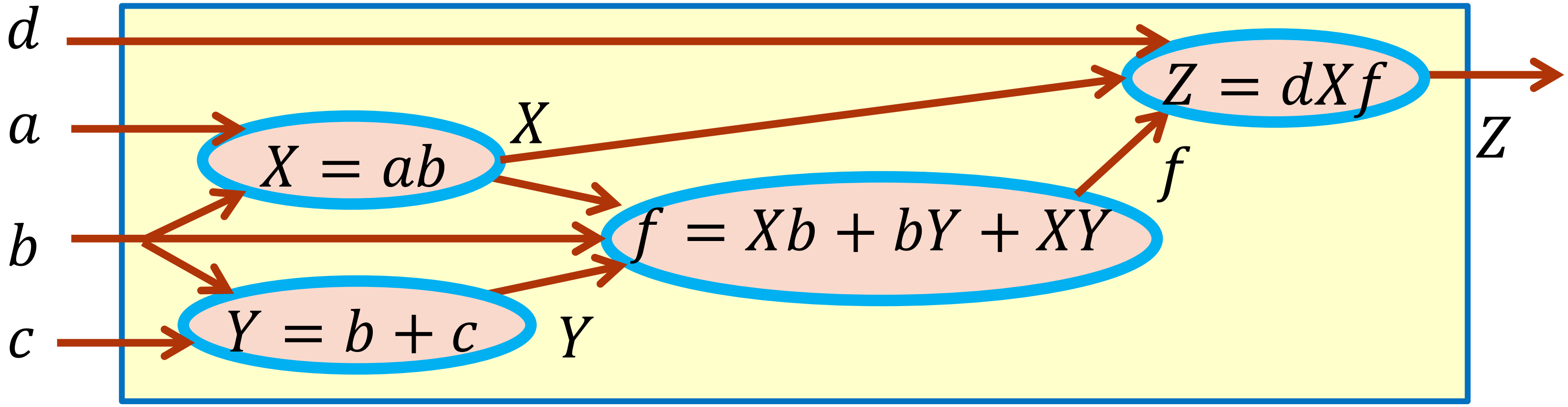
- **Example**
  - For node f, $(X, b, Y) = (1,0,0), (1,0,1)$ are CDCs.



Observability don’t cares: ODCs

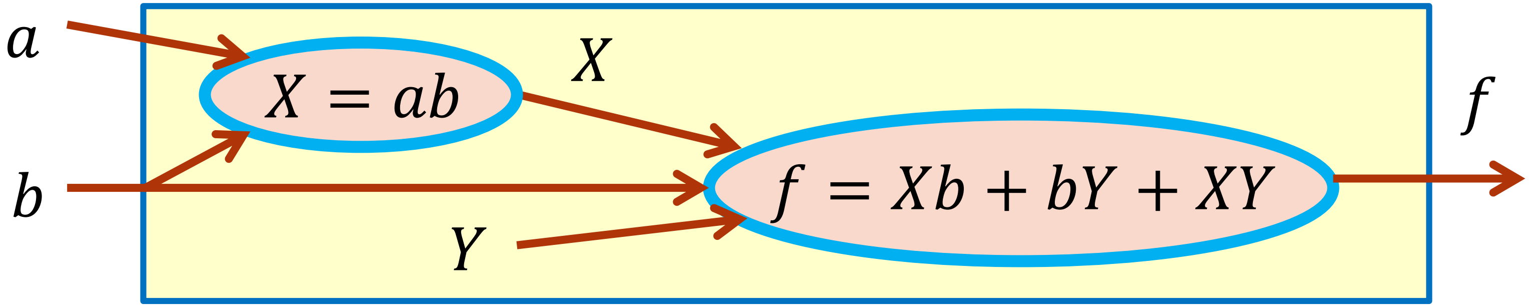
- Input patterns to node that make primary outputs **insensitive** to output of the node.
    - Patterns that “**mask**” outputs.

- **Example**
    - For node f, $(X, b, Y) = (0, -, -)$ is ODC.



Background: Representing DC Patterns

- How shall we **represent** DC patterns at a node?
  - **Answer**: As a **Boolean function** that makes a 1 when the inputs are **these DCs**.
  - This is often called a **Don’t Care Cover**.



Don’t care pattern of (X,b,Y)=(1,0,0), (1,0,1)

The don’t care cover is $X\bar{b}\bar{Y} + X\bar{b}Y = X\bar{b}$

Background: Representing DC Patterns

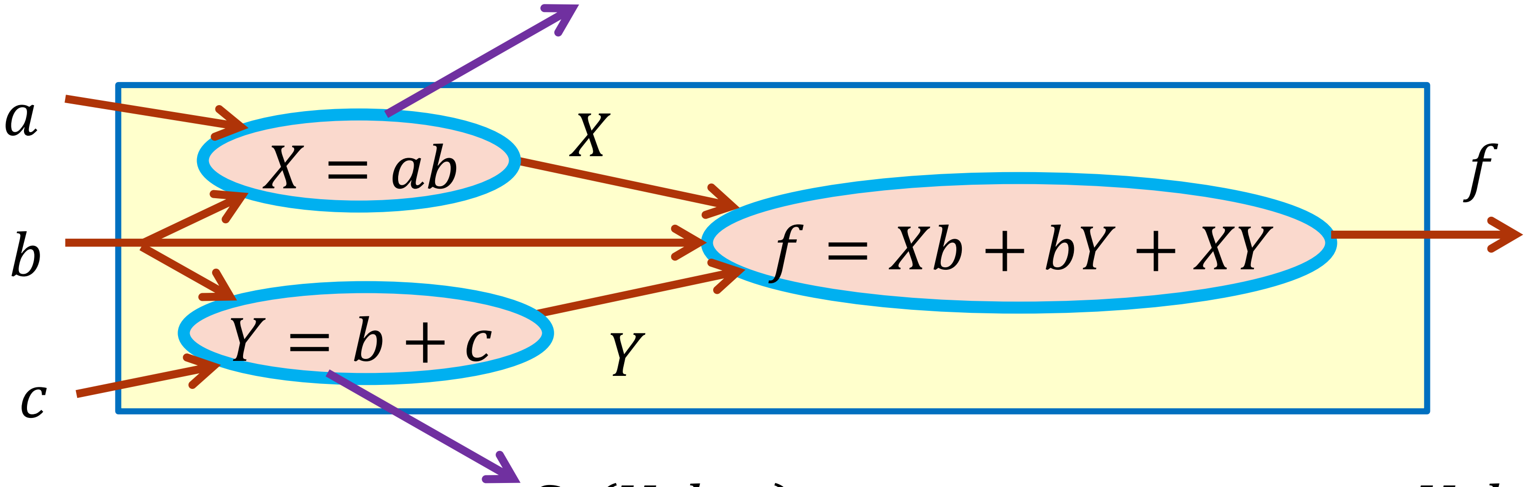
- So, each SDC, CDC, ODC is just another Boolean function, in this strategy.

- Why is it like this?
  - Because we can use all the other **computational Boolean algebra** techniques we know (e.g., BDDs), to **solve** for, and to **manipulate** the DC patterns.
  - This turns out to be hugely important to make the computation practical.

SDCs: They “Belong” to the Wires

- One SDC for every **internal wire** in Boolean logic network.
  - The SDC represents **impossible** patterns of **inputs to, and output of**, each node.
  - If the node function is F, with inputs a, b, c, write its SDC as: $S_F(F, a, b, c)$.

$S_X(X, a, b)$ for impossible patterns of X, a, b.

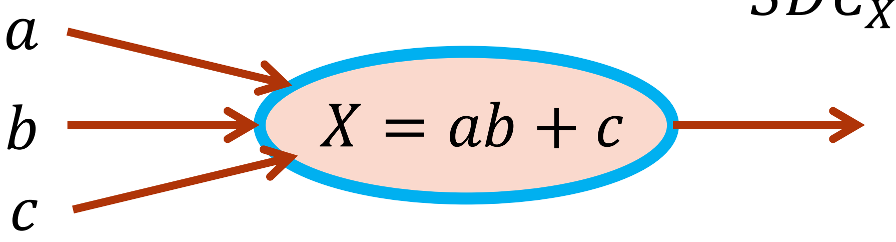


$S_Y(Y, b, c)$ for impossible patterns of Y, b, c.

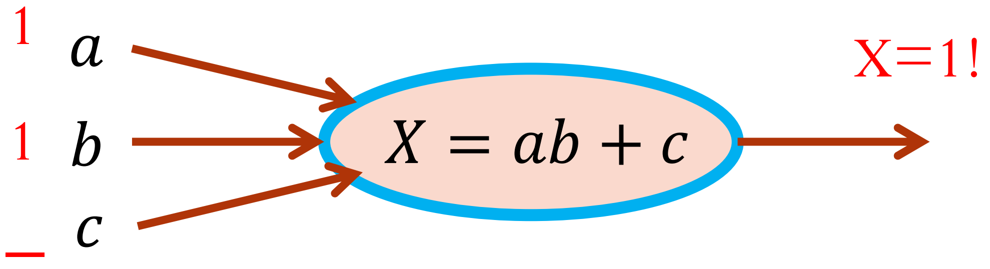
SDCs: How to Compute

- Compute an SDC for each output wire from each internal Boolean node.
- You want an expression that is 1 when output X **does not equal** the Boolean expression for X.
  - This is just: $X \oplus$ (expression for X)
    - **Note #1**: expression for X doesn't have X in it!
    - **Note #2**: this is the **complement** of the gate consistency function from SAT.
- Example

$$SDC_X = X \oplus (ab + c)$$



SDCs: Example



- $SDC_X = X \oplus (ab + c) = \boxed{\bar{X}ab} + \bar{X}c + X\bar{a}\bar{c} + X\bar{b}\bar{c}$

⬇

One **impossible pattern**: $Xabc = 011 -$

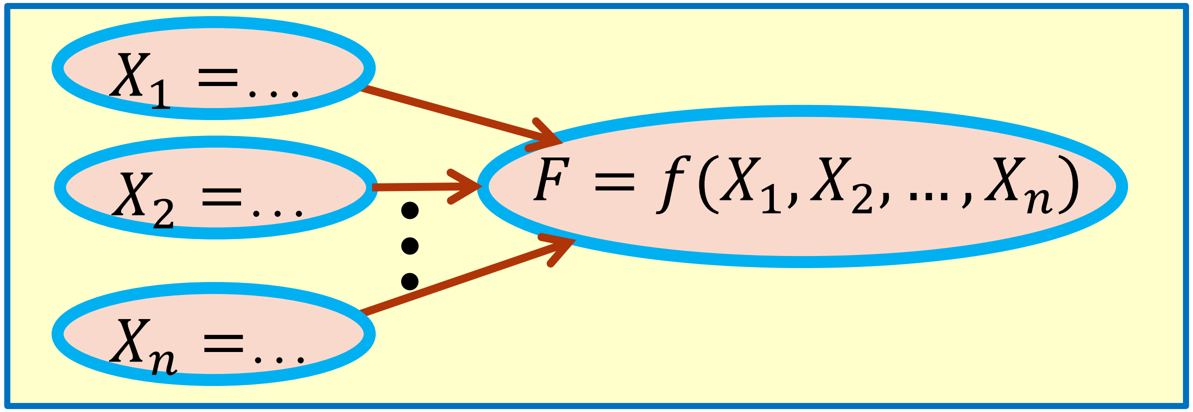
SDCs: Summary

- SDCs are associated with every **internal wire** in Boolean logic network.
  - SDCs explain **impossible patterns** of input to, and output of, each node.
  - SDCs are easy to compute.

- SDCs alone are **not** the Don't Cares used to simplify nodes.
  - We use SDCs to **build CDCs**, which give impossible patterns at input of nodes.

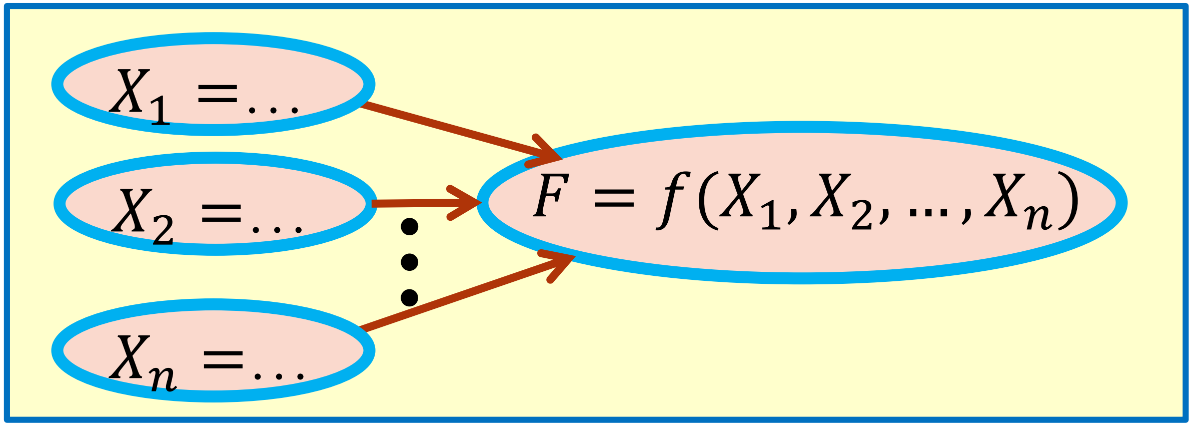
How to Compute CDCs?

- Computational recipe:
  1. Get all the **SDCs** on the wires **input to** this node in Boolean logic network.
  2. **OR** together all these SDCs.
  3. **Universally Quantify** away all variables that are **NOT** used inside this node.



$$CDC_F(X_1, \ldots, X_n) = (\forall \text{ vars not used in } F)\left[\sum_{\text{input } X_i \text{ to } F} SDC_{X_i}\right]$$

How to Compute CDCs?



$$CDC_F(X_1, \ldots, X_n) = (\forall \text{ vars not used in } F)\left[\sum_{\text{input } X_i \text{ to } F} SDC_{X_i}\right]$$

- **Result**: Inputs that let $CDC_F = 1$ are **impossible patterns** at input to node!

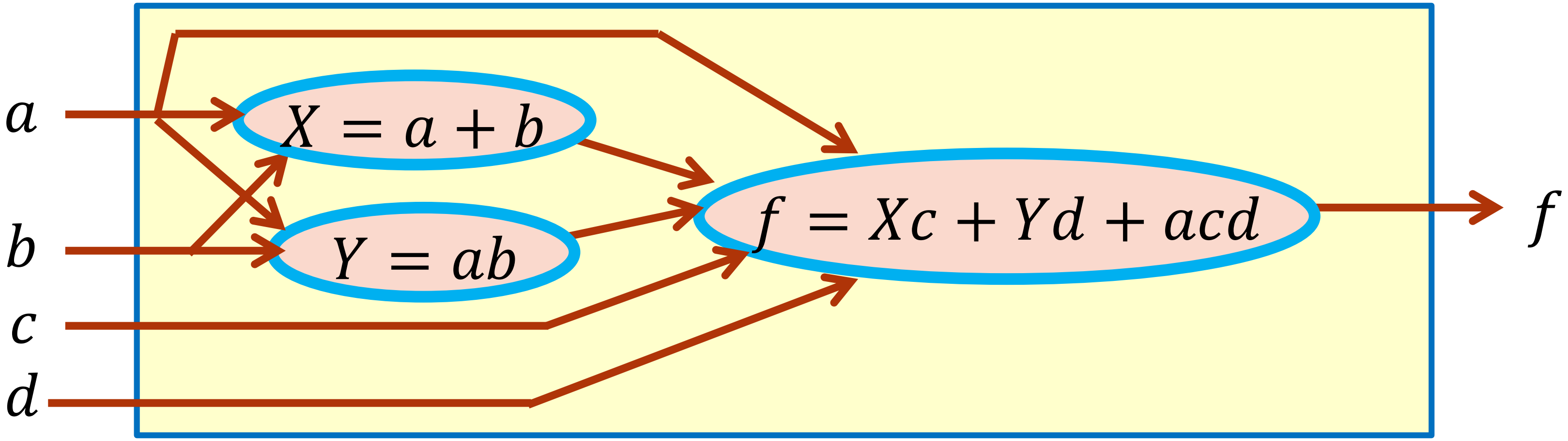
CDCs: Why Does This Work?

$$CDC_F(X_1, \ldots, X_n) = (\forall \text{ vars } \text{not used} \text{ in } F)\left[\sum_{\text{input } X_i \text{ to } F} SDC_{X_i}\right]$$

- Roughly speaking…
  - SDC_{X_i}'s explain all the impossible patterns involving X_i wire input to the F node.
  - **OR** operation is just the "**union**" of all these impossible patterns involving X_i's.
  - **Universal Quantification** removes variables **not** used by F, and does so in the right way: we want patterns that are impossible **FOR ALL** values of these removed variables.

Compute CDCs: Example

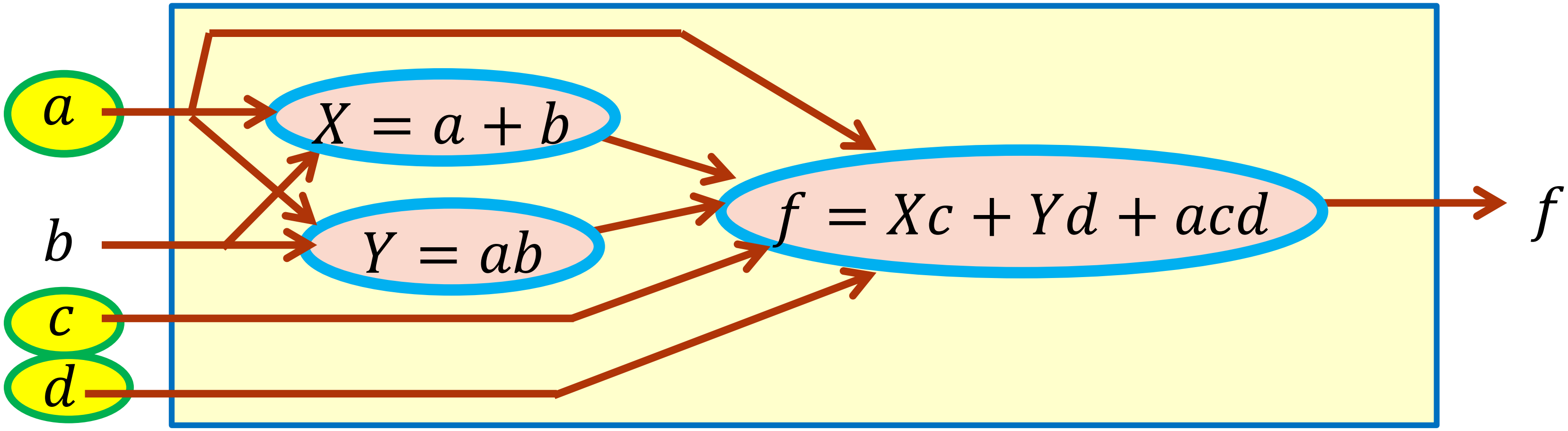
Obtain CDCs for the node f



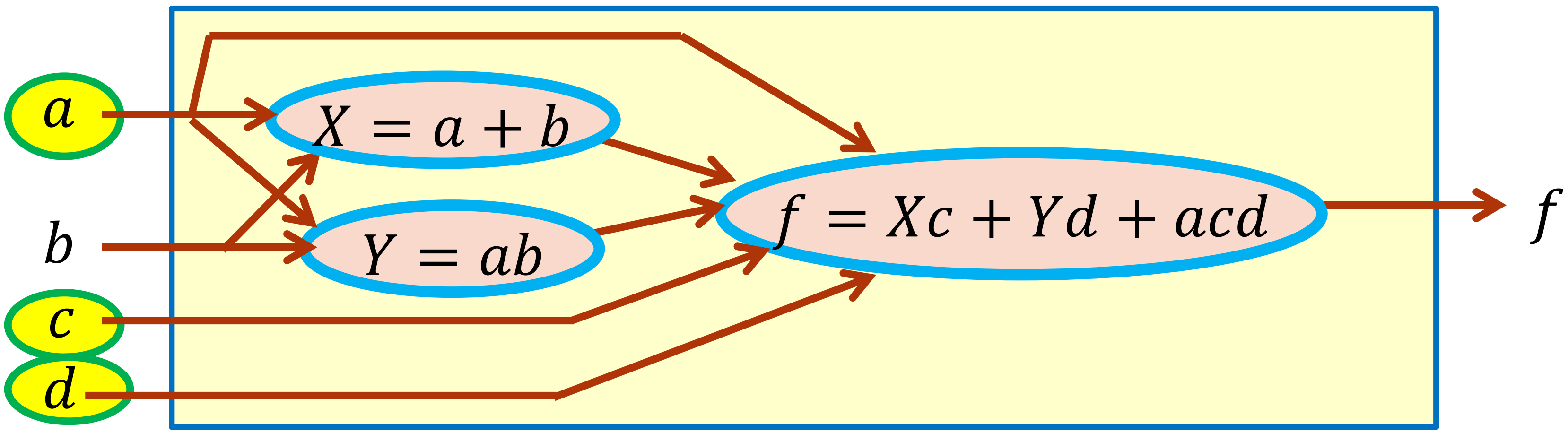
$$CDC_f(X_1, \dots, X_n) = \underbrace{(\forall \text{ vars } \text{not used} \text{ in } f)}_{\text{This is } b} \underbrace{\left[\sum_{\text{input } X_i \text{ to } f} SDC_{X_i}\right]}_{\text{Input variables to } f \text{ are } a, c, d, X, Y}$$

Compute CDCs: Example

- What about SDCs on **primary inputs**?
  - They are just 0.
  - Why? $SDC_a = a \oplus (\text{expression for } a) = a \oplus a = 0$.
- **Thus**: SDCs on primary inputs have no impact on OR. We can **ignore primary inputs**.



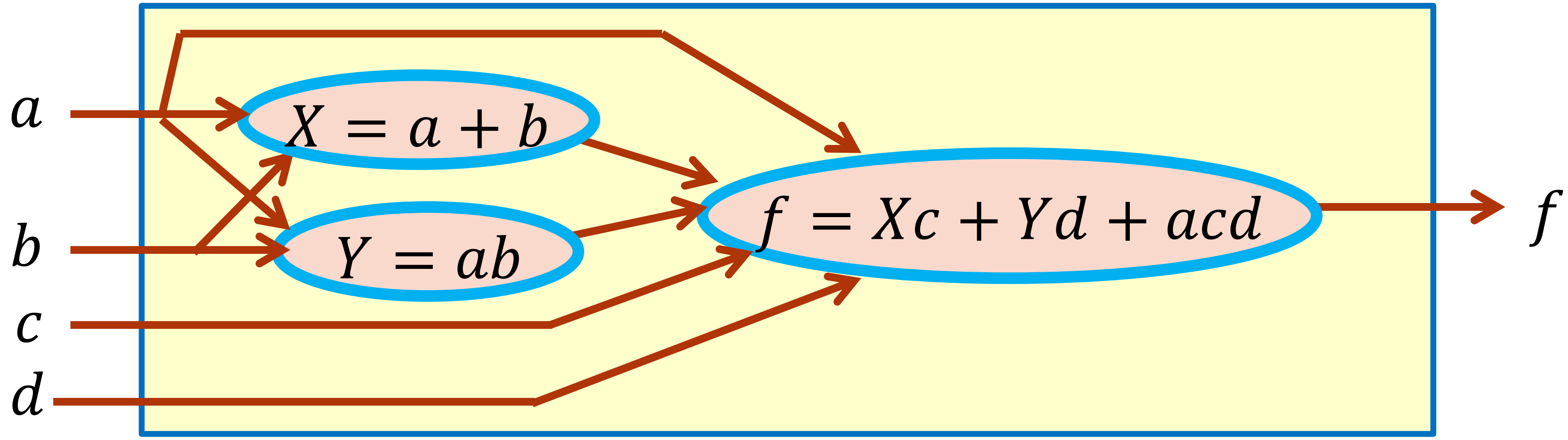
Compute CDCs: Example



- Since we ignore primary inputs, we have …

$$CDC_f(X_1, \ldots, X_n) = \underbrace{(\forall \text{ vars not used in } f)}_{\text{This is } b} \left[\underbrace{\sum_{\text{input } X_i \text{ to } f}}_{\boxed{\text{Only } X, Y}} SDC_{X_i} \right]$$

Compute CDCs: Example



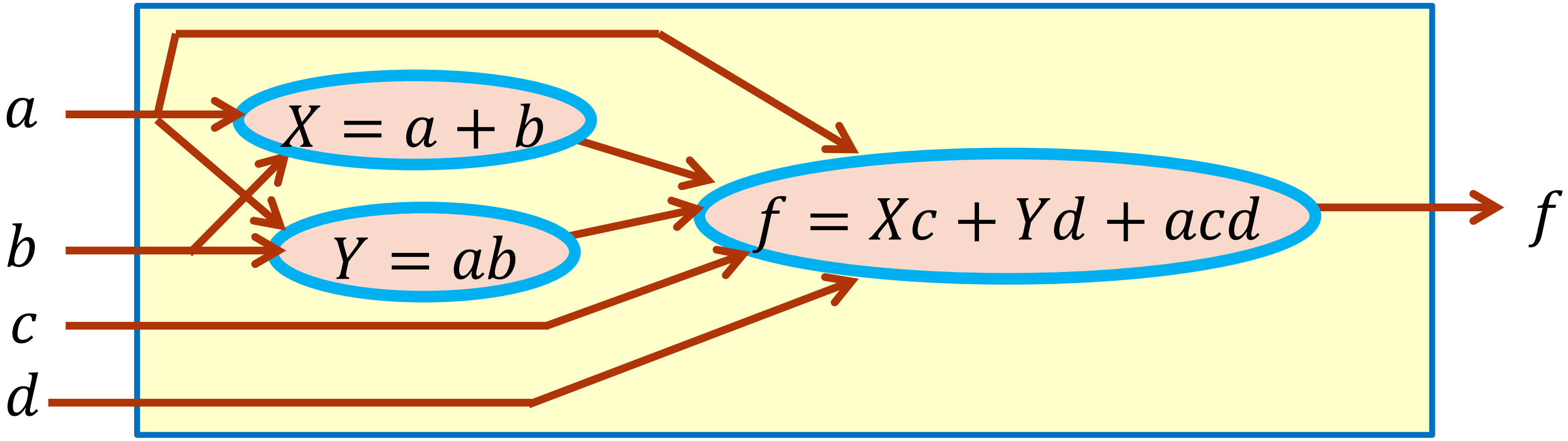
- Thus, we have:

$$CDC_f = (\forall b)[SDC_X + SDC_Y] = (\forall b)\big[[X\oplus(a+b)] + [Y\oplus ab]\big]$$

$$= \big[[X\oplus(a+b)] + [Y\oplus ab]\big]_{b=1} \cdot \big[[X\oplus(a+b)] + [Y\oplus ab]\big]_{b=0}$$

$$= [\bar{X} + (Y\oplus a)] \cdot [(X\oplus a) + Y] = \bar{X}a + Y\bar{a}$$

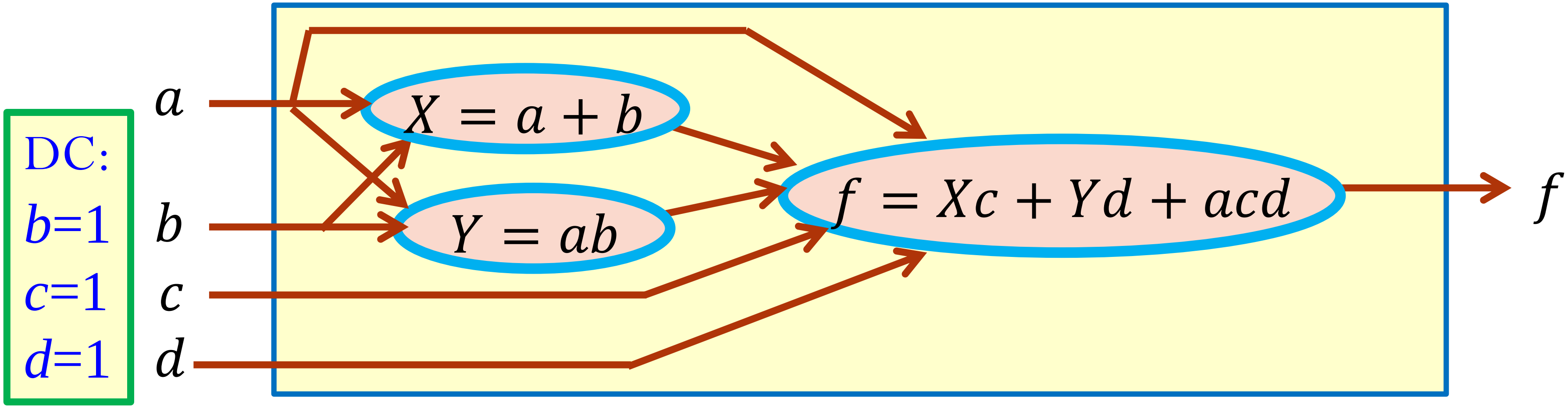
Compute CDCs: Example



- $CDC_f = \bar{X}a + Y\bar{a}$
- Does it **make sense**?
  - From CDC_f, **impossible patterns** are
    - $(X, a) = (0,1)$ $a = 1 \Rightarrow X = 1$
    - $(Y, a) = (1,0)$ $a = 0 \Rightarrow Y = 0$

How to Handle External CDCs?

- What if there are **external DCs** for primary inputs a, b, c, d for which we just **don't care** what f does?
  - **Answer**: Just **OR** these DCs in $(\sum SDC_i)$ part of CDC expression.
  - Represent these DCs as a **Boolean function** that makes a 1 when the inputs are **these DCs**.



DC:

b=1

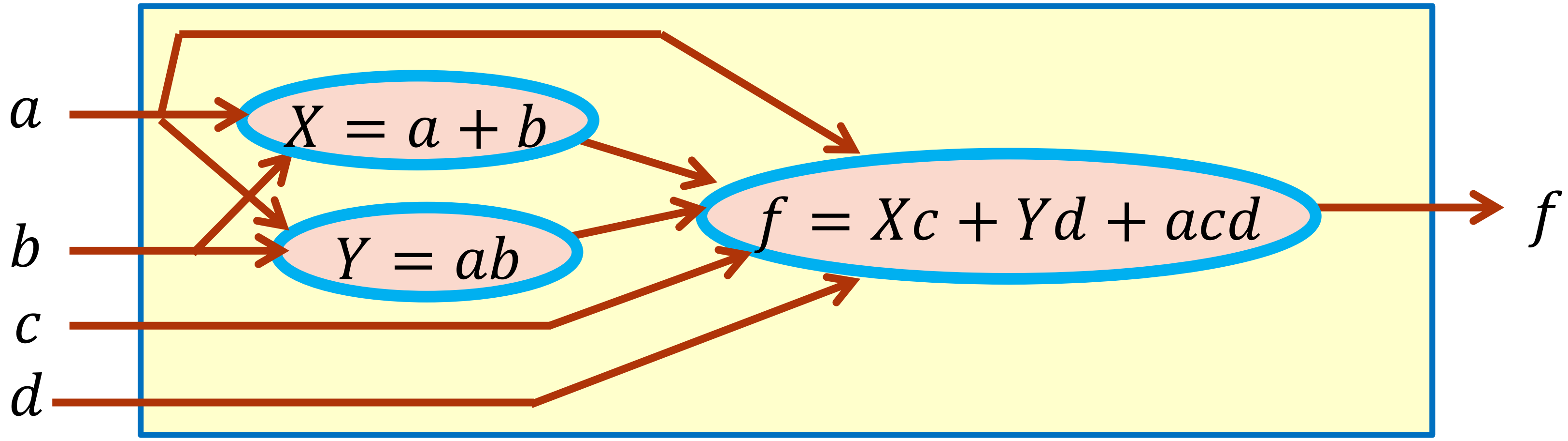
c=1

d=1

Handling External CDCs: Example

- Suppose $(b, c, d) = (1,1,1)$ cannot happen.
  - How to compute CDC_f now?

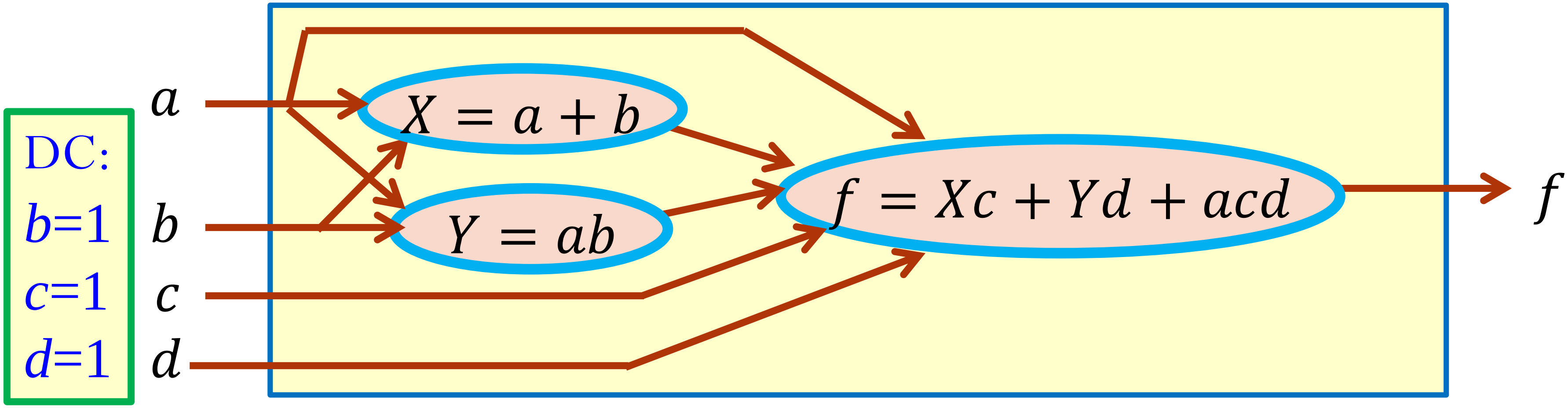
DC:
b=1
c=1
d=1



$$CDC_f = (\forall b)\Big[[X\oplus(a+b)] + [Y\oplus ab] + \underbrace{bcd}\Big]$$

External DCs as a **Boolean function** that makes a 1 when the pattern is **impossible**.

Handling External CDCs: Example



$$CDC_f = (\forall b)\big[[X \oplus (a+b)] + [Y \oplus ab] + bcd\big]$$

$$= \bar{X}a + Y\bar{a} + \bar{a}cdX + cdY$$

- **New impossible patterns** are

Make sense?

  - $(a, c, d, X) = (0,1,1,1)$ $a = 0 \,\&\&\, X = 1 \Rightarrow b = 1$

    Thus, $b = c = d = 1$
  - $(c, d, Y) = (1,1,1)$ $Y = 1 \Rightarrow b = 1$

    Thus, $b = c = d = 1$

CDCs: Summary

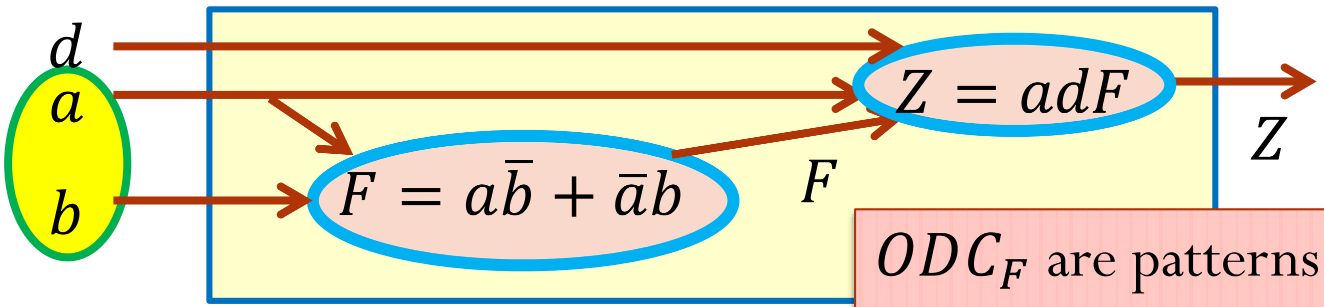
- CDCs give **impossible patterns** at input to node F – use as DCs.
  - Impossible because of the network structure of the nodes **feeding** node F.
  - CDCs can be computed mechanically from **SDCs** on wires input to F.
    - **Internal local CDCs**: computed just from SDCs on wires into F.
    - **External global CDCs**: include DC patterns in the SDC sum.

CDCs: Summary (cont.)

- But CDCs are still **not all** the Don’t Cares available to simplify nodes.
  - CDC_F derived from the structure of nodes “**before**” node F.
  - We need to look at DCs that derive form nodes “**after**” node F.
  - These are nodes between the **output** of F and **primary outputs** of the network.
  - These are ODCs.

Observability Don’t Cares (ODCs)

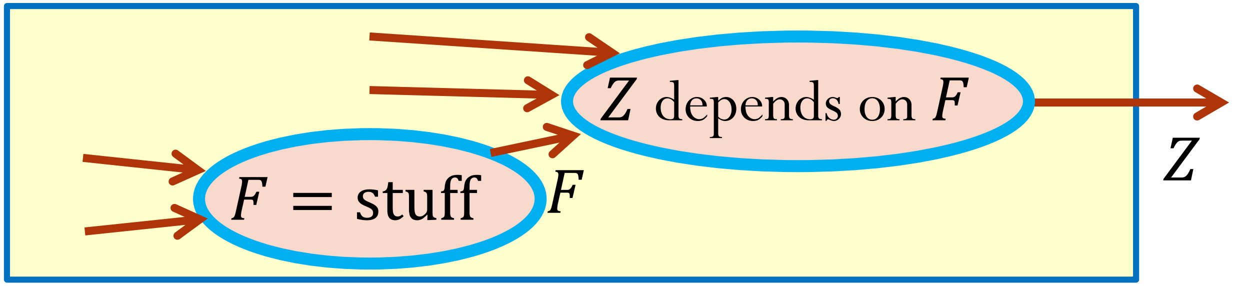
- **ODCs**: patterns that **mask** a node’s output at primary output (PO) of the network.
  - So, these are **not** impossible patterns – these patterns **can occur** at node input.
  - These patterns make this node’s output **not observable at primary output**.
  - “**Not observable**” for an input pattern means: Boolean value of node output **does not affect** **<u>ANY</u>** primary output.

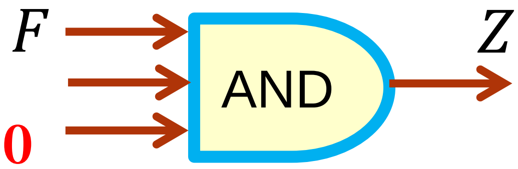


ODC_F are patterns of (a, b) that make Z **insensitive** to F’s value.

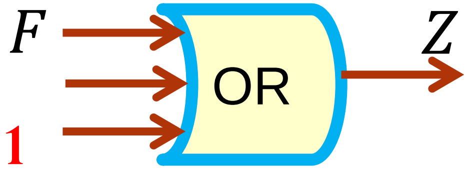
Primary Output Insensitive to F

- When is primary output Z **insensitive** to internal variable F?
  - Means Z **independent** of value of F, given other inputs to Z.





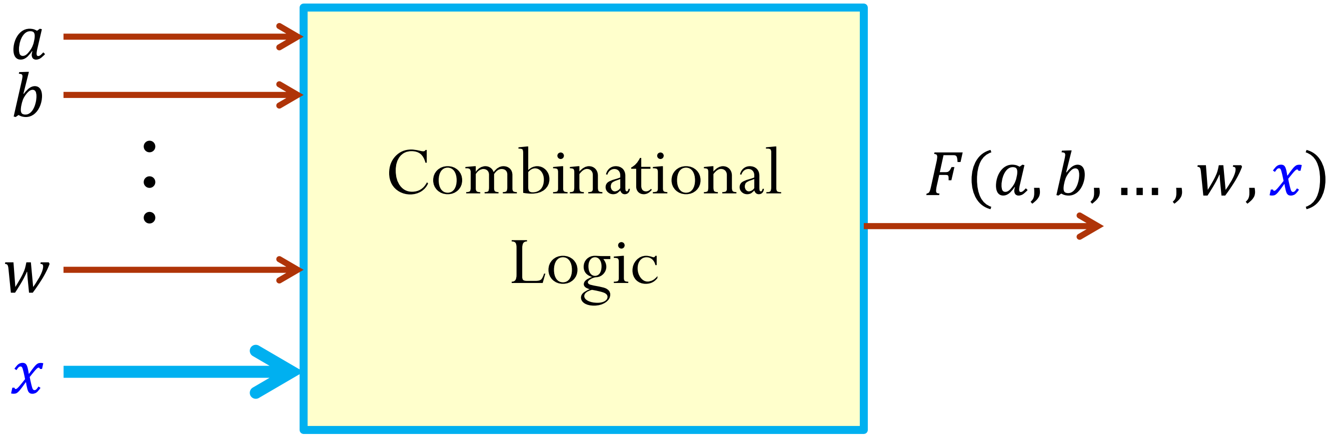
Z **insensitive** to F if **another** input = 0



Z **insensitive** to F if **another** input = 1

How about the general case?

Recall: Boolean Difference



- What does **Boolean difference** $\partial F(a, b, \ldots, w, x)/\partial x = F_x \oplus F_{\overline{x}} = 1$ mean?
    - If you apply an input pattern $(a, b, \ldots, w)$ that makes $\partial F/\partial x = 1$, then **any change** in x will **force a change** in output F.
- What makes output F **sensitive** to input x?
    - **Answer**: Any pattern that makes $\frac{\partial F}{\partial x} = F_x \oplus F_{\overline{x}} = 1$.

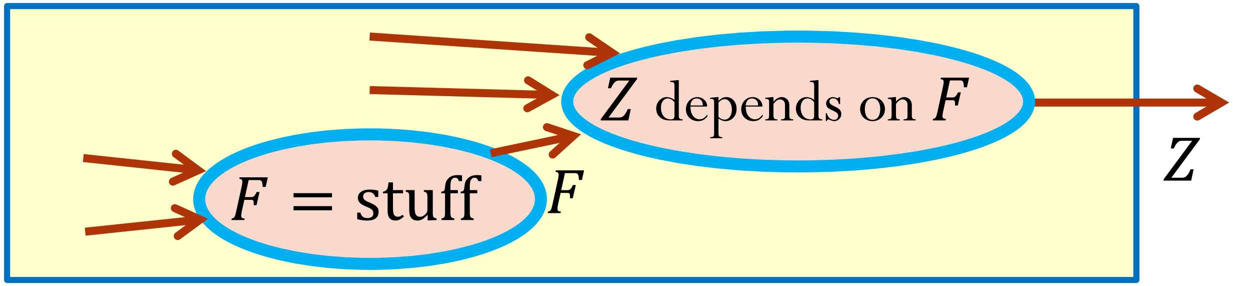
Z Insensitive to F

- When is primary output Z **insensitive** to internal variable F?
  - **Answer**: when inputs (other than F) to Z make cofactors $Z_F = Z_{\bar{F}}$.
  - **Make sense**: if cofactors with respect to F are **same**, Z does not depend on F!

- How to find when cofactors are the same?
  - **Answer**: Solve for $Z_F \overline{\oplus} Z_{\bar{F}} = 1$
  - **Note**: $Z_F \overline{\oplus} Z_{\bar{F}} = 1 \Rightarrow \overline{Z_F \oplus Z_{\bar{F}}} = 1 \Rightarrow \overline{\frac{\partial Z}{\partial F}} = 1$

How to Compute ODCs?

- A nice computational recipe:
  1. **Compute** $\overline{\partial Z/\partial F}$. Any patterns that make $\overline{\partial Z/\partial F} = 1$ **mask** output F for Z.
  2. **Universally Quantify** away all variables that are **NOT** inputs to the F node.

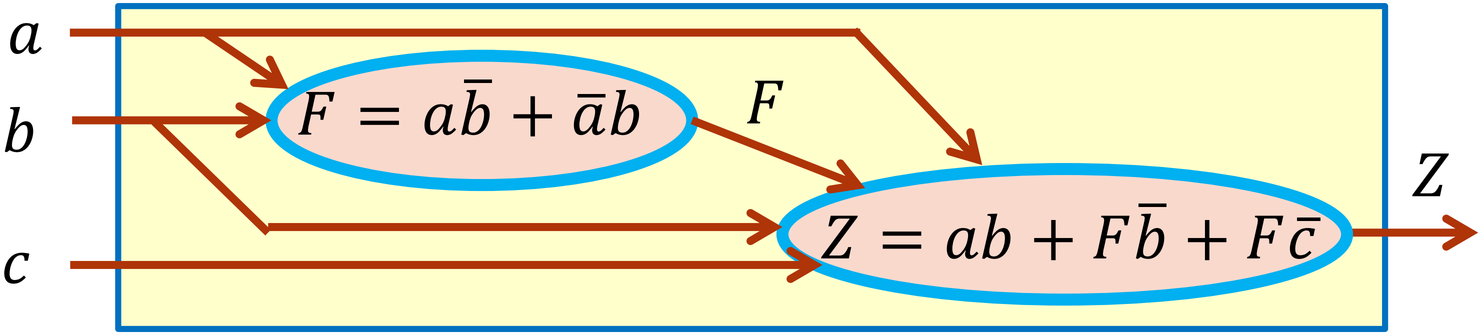


$$ODC_F(X_1, \ldots, X_n) = (\forall \text{ vars not used in } F) \left[\overline{\partial Z/\partial F}\right]$$

<u>**Result**</u>: Inputs that let $ODC_F = 1$ **mask** output F for Z, i.e., make Z **insensitive** to F.

Compute ODCs: Example

- Obtain the ODCs for node F.

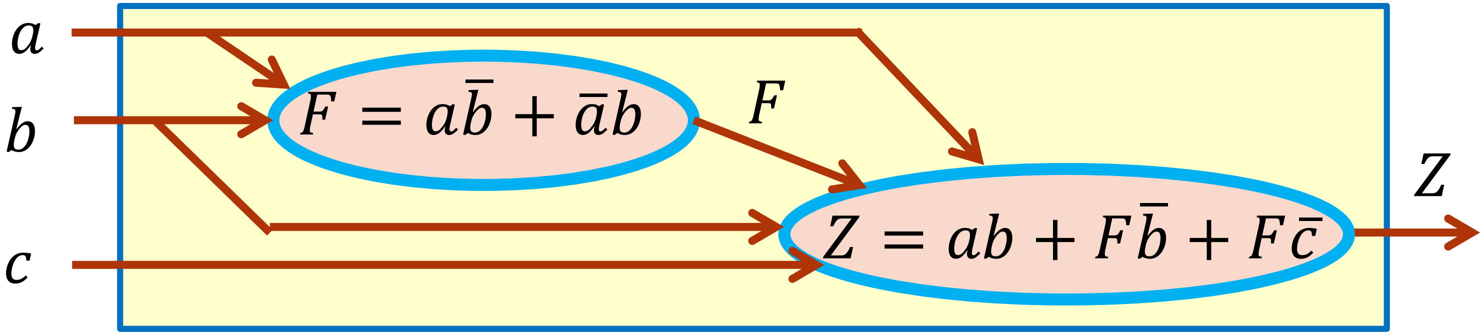


$$ODC_F\underbrace{(X_1, \ldots, X_n)}_{\text{They are } a,\, b} = \underbrace{(\forall \text{ vars not used in } F)}_{\text{This is } c}\left[\overline{\partial Z/\partial F}\right]$$

$$= (\forall c)\left[\overline{\left(ab + F\bar{b} + F\bar{c}\right)_{F=1} \oplus \left(ab + F\bar{b} + F\bar{c}\right)_{F=0}}\right]$$

$$= (\forall c)\left[\overline{(a + \bar{b} + \bar{c}) \oplus (ab)}\right] = ab$$

Check: Does this ODC Make Sense?

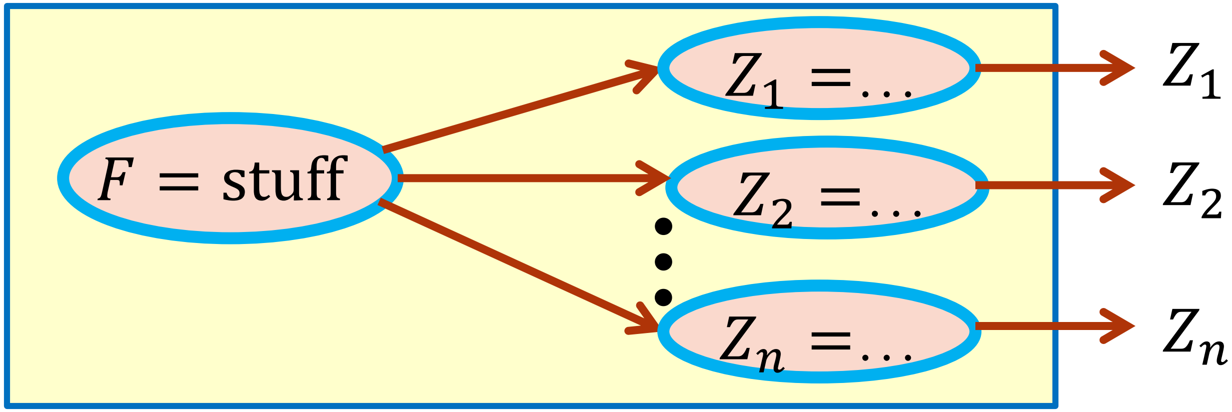


- $ODC_F = ab$
  - ODC pattern is $(a, b) = (1,1)$

- Make sense! Because when $(a, b) = (1,1), Z = 1$ independent of F.

ODCs: More General Case

- **Question**: what if F feeds to **many** primary outputs?
  - **Answer**: Only patterns that are **unobservable** at **ALL** outputs can be ODCs.



- Computational recipe:

$$ODC_F = (\forall \text{ vars not used in } F)\underbrace{\left[\prod_{\text{Output } Z_i} \overline{\partial Z_i / \partial F}\right]}_{}$$

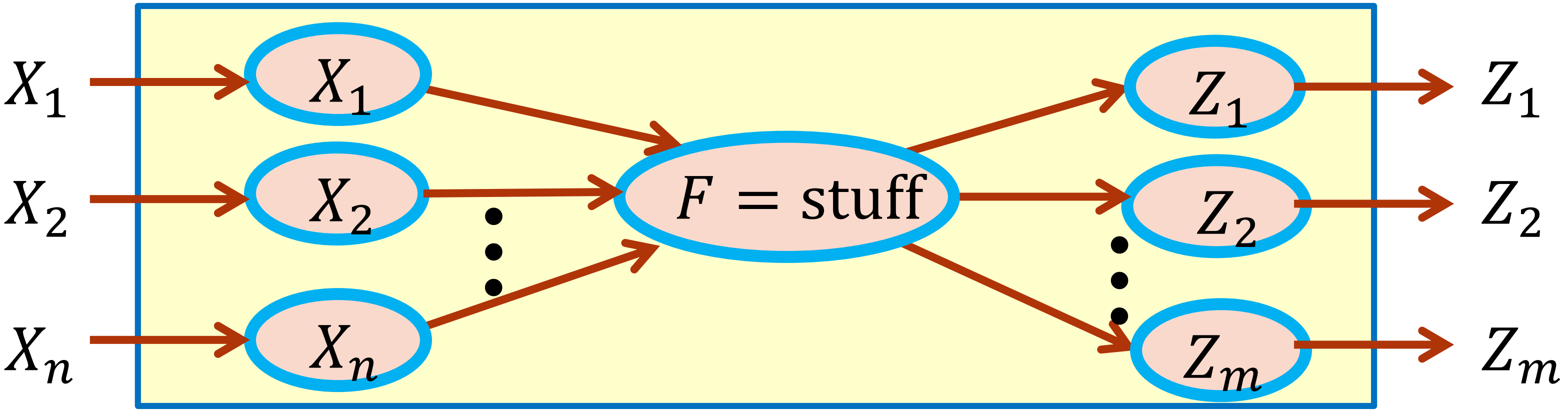
AND all n differences for each output Z_i.

ODCs: Summary

- ODCs give input patterns of node F that **mask** F at **primary outputs**.
  - **Not** impossible patterns – they **can occur**.
  - Don't cares because primary output "**doesn't care**" what F is, for these patterns.
  - ODCs can be computed mechanically from $\overline{\partial Z_i / \partial F}$ on all outputs connected to F.

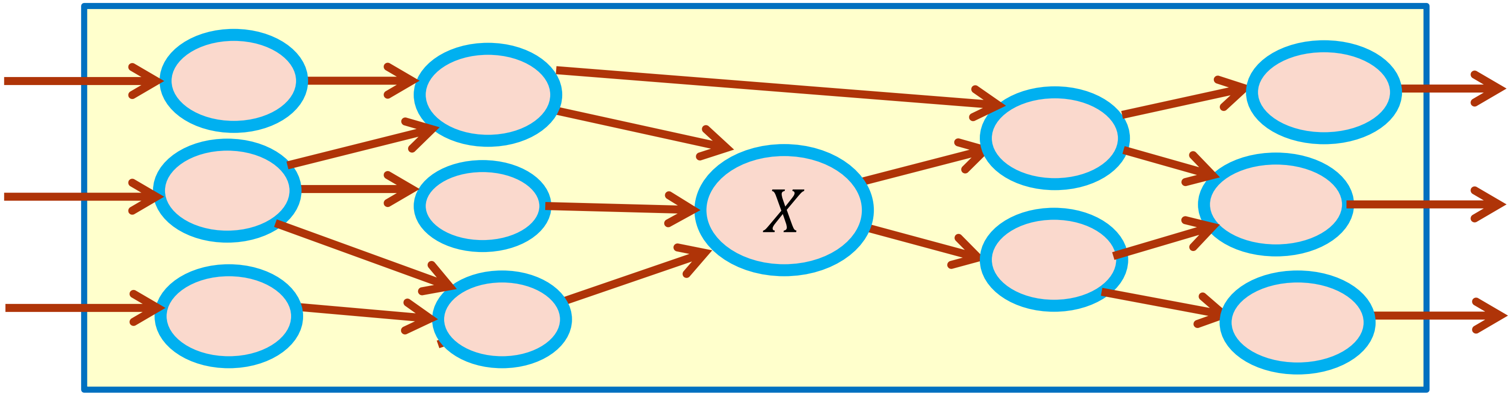
- CDCs + ODCs give the "**full**" don't care set used to simplify F.
  - With these patterns, you can call something like ESPRESSO to simplify F.

Multi-Level Don’t Cares: Are We Done?



- Yes, if your networks look **just like above**.
  - More precisely, if you only want to get CDCs from nodes **immediately** “before” you.
  - And if you only want to get ODCs for **one layer of nodes** between you and output.

Don’t Cares, In General



- However, **real** multi-level logic looks like this!
  - CDCs are function of **all nodes** “before” *X*.
  - ODCs are function of **all nodes** between *X* and any output.
  - In general, we can **never get all** the DCs for node *X* in a big network.
  - Representing all this stuff can be **explosively** large, even with BDDs

Summary: Getting Network DCs

- How we really do it? generally **do not get all** the DCs.
  - Lots of tricks that trade off effort (time, memory) with quality (how many DCs).
  - Example: Can just extract "**local CDCs**", which requires looking at outputs of **immediate precedent** vertices and computing from the SDC patterns, which is easy.
  - There are also algorithms that walk the network to compute more of the CDC and ODC set for X, but these are more **complex**.

- For us, knowing these "limited" DC recipes is **sufficient**.