

CSE 620: Selective Topics

Introduction to Formal Verification



Master Studies in CSE
Fall 2016
Lecture #1



Dr. Hazem Ibrahim Shehata

Assistant Professor

Dept. of Computer & Systems Engineering



Course Info

- Instructor

- Hazem Ibrahim Shehata
- Email: hshehata@uwaterloo.ca
- Lectures: Monday 11:00am-1:00pm

- Course website

hshehata.github.io/courses/zu/cse620

- Grading

- Total mark (100) = **Assignments** (30) + **Final** (70).
- **Three** programming assignments are required.
- For each assignment, each student needs to:
 1. Submit a hard-copy of his/her **report** (3-page long at most).
 2. Submit a soft-copy of his/her **code** (".zip" file by email).
 3. Demo his/her running **code** to the instructor.



Evaluation Techniques

Validation

- Goals:
 - Are we building the right thing?!
 - Does product satisfy actual user needs?
 - Acceptance/Suitability
- Takes place at the end of development
- External process
- Dynamic

Verification

- Goals:
 - Are we building it right?!!
 - Does product conform to specifications?
 - Correctness
- Takes place during development (between phases)
- Internal process
- Dynamic or Static

Verification Techniques

Informal Verification

- Dynamic
 - Simulation-based
- Uses a test bench
 - Stimulus & monitor
- Partial coverage
 - Some input/state combinations
- Requires no formal specification
- No guarantee for correctness

Formal Verification

- Static
- Proves/disproves design correctness
- Complete coverage
 - All input/state combinations
- Requires formal specification
- Guarantees conformance to specification



Formal Verification Techniques

Theorem Proving

- Verification by proving a theorem
- Deductive
- Human-guided
- **Model**: set of logical statements
- **Specification**: set of logical statements
- Requires an expert!

Model Checking

- Verification through exhaustive search
- Algorithmic
- Automatic
- **Model**: state machine (explicit or symbolic).
- **Specification**: high-level model or properties
- Doesn't scale well!
 - state-space explosion



Course Goal, Focus and Outline

- Goal

- To introduce you to the formal verification field

- Focus

- Automatic techniques → model checking

- Outline

- Computational Boolean Algebra

- Basics
- Validity Checking (Tautology Checking)
- Satisfiability Checking (SAT solving)
- Binary Decision Diagrams (BDD's)

- Model Checking

- Temporal Logics → LTL - CTL
- SMV: Symbolic Model Verifier
- Model Checking Algorithms → Explicit CTL





VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

Lecture 2.1

Computational Boolean
Algebra: Basics



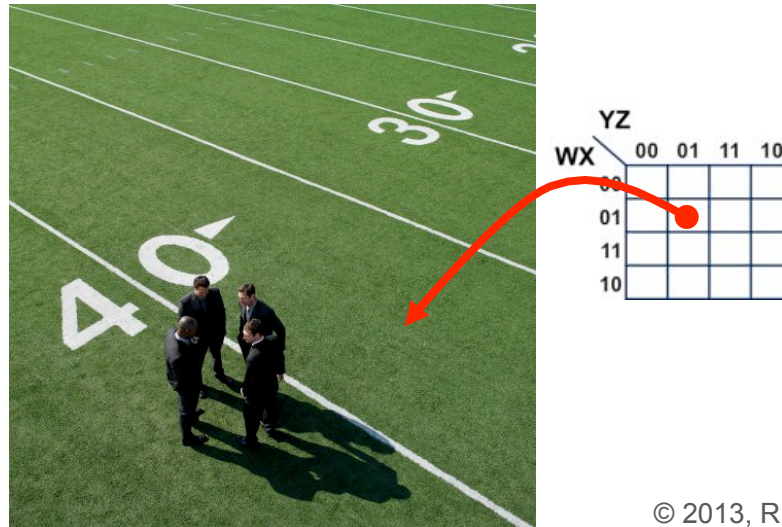
Computational Boolean Algebra...?

- **Background...**

- You've done Boolean algebra, hand manipulations, Karnaugh maps to simplify...
- But this is **not sufficient** for real designs

- **Example: simplify Boolean function of 40 variables via Kmap**

- It has **1,099,511,627,776** squares
- You could fit this on an American style football field...
- ...but each Kmap square would be just **60 x 60** microns!
- There must be a **better** way...



Need a Computational Approach

- Need **algorithmic, computational** strategies for Boolean stuff
 - Need to be able to think of Boolean objects as **data structures + operators**
- What will we study?
 - **Decomposition strategies**
 - Ways of taking apart complex functions into simpler pieces
 - A set of advanced concepts, terms you need to be able to do this
 - **Computational strategies**
 - Ways to think about Boolean functions that let them be manipulated by programs
 - **Interesting applications**
 - When you have new tools, there are some useful new things to do



Advanced Boolean Algebra

- **Useful analogy to calculus.** e^x
 - You can represent complex functions like **exp(x)** using simpler functions
 - If you only get to use 1, x, x², x³, x⁴, ... as the pieces...
 - ...turns out **exp(x) = 1 + x + x²/2! + x³/3! + ...**
 - In Calculus, we tell you the general formula, the **Taylor series expansion**
 - **$f(x) = f(0) + f'(0)/1! x + f''(0)/2! x^2 + f'''(0)/3! x^3 + ...$**
 - If you take more math, you might find out several other ways:
 - If it's a periodic function, can use a **Fourier series**
- **Question: Anything like this for Boolean functions?**



Boolean Decompositions

- Yes. Called the *Shannon Expansion*

The top screenshot shows the Wikipedia article for Claude Shannon. The title 'Claude Shannon' is at the top. Below it, the text reads: 'Claude Elwood Shannon (April 30, 1916 – February 24, 2001) was an American electronic engineer, and cryptographer known as the father of Information Theory'. A blue circle is drawn around the text 'Shannon's expansion' in the first paragraph of the article.

The bottom screenshot shows the Wikipedia article for Shannon's expansion. The title 'Shannon's expansion' is at the top. Below it, the text reads: 'In mathematics, **Shannon's expansion** or the **Shannon decomposition** is a method by which a Boolean function can be represented by the sum of two sub-functions of the original. Although it is often credited to Claude Shannon, Boole proved this much earlier. Shannon is credited with many other important aspects of Boolean algebra'. A blue circle is drawn around the title 'Shannon's expansion'.

Slide 5

Shannon Expansion

- Suppose we have a function $F(x_1, x_2, \dots, x_n)$
- Define a **new function** if we set one of the $x_i = \text{constant}$
 - Example: $F(x_1, x_2, \dots, x_i=1, \dots, x_n)$
 - Example: $F(x_1, x_2, \dots, x_i=0, \dots, x_n)$
- Easy to do one by hand

$$F(x, y, z) = xy + xz' + y(x'z + z')$$

$$F(\underline{x=1}, y, z) = y\bar{z} + y\bar{z}$$

$$F(x, \underline{y=0}, z) = 0 + x\bar{z} + 0 = x\bar{z}$$

- Note: this is a **new function**, that no longer depends on this variable (**var**)

Shannon Expansion: Cofactors

- **Turns out to be an incredibly useful idea**

- Several alternative names and notations
- Shannon Cofactor with respect to x_i

- Write $F(x_1, x_2, \dots, \text{xi=1}, \dots, x_n)$ as: F_{x_i} = positive cofactor
- Write $F(x_1, x_2, \dots, \text{xi=0}, \dots, x_n)$ as: $F_{\bar{x}_i}$ - negative cofactor

- Often write this as just $F(\text{xi=1})$ $F(\text{xi=0})$ which is easier to type

- **Why are these useful functions to get from F ?**



Shannon Expansion Theorem

- **Why we care: Shannon Expansion Theorem**

- Given any Boolean function $F(x_1, x_2, \dots, x_n)$ and pick any x_i in $F()$'s inputs
 $F()$ can be represented as

$$F(x_1, x_2, \dots, \mathbf{x_i}, \dots, x_n) = \mathbf{x_i} \cdot \overset{\text{pos}}{F(\mathbf{x_i}=1)} + \mathbf{x_i'} \cdot \overset{\text{neg}}{F(\mathbf{x_i}=0)}$$

- Pretty easy to prove...

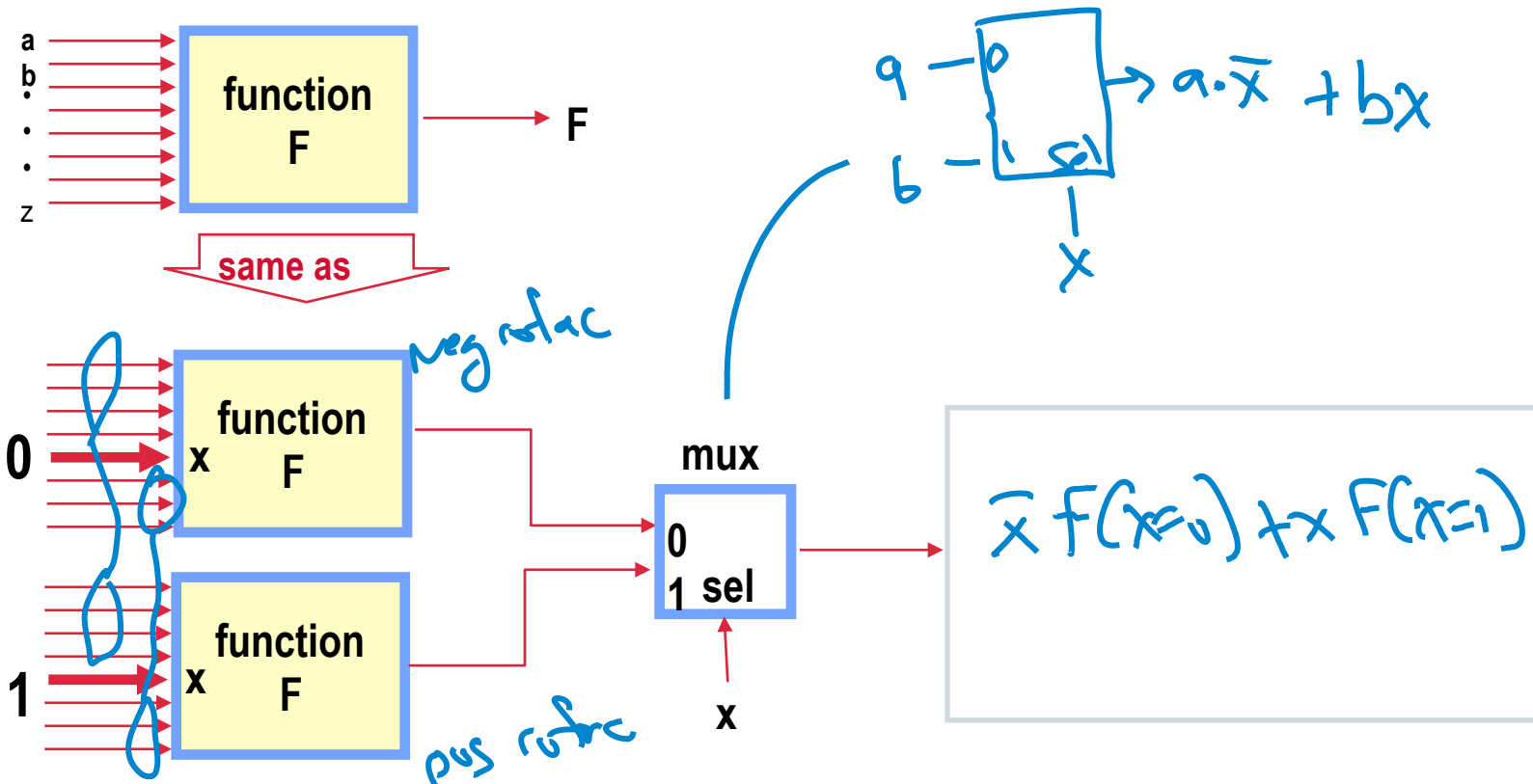
Let $x_i = 1$

$$\begin{aligned} F(\dots, x_i=1, \dots) &= 1 \cdot F(x_i=1) \\ &\quad + \cancel{0 \cdot F(x_i=0)} \\ &= F(x_i=1) \quad \checkmark \end{aligned}$$

Let $x_i = 0$

same
↓

Shannon Expansion: Another View



Shannon Expansion: Multiple Variables

- Can do it on *more than one* variable, too

- Just keep on applying the theorem

- Example $F(x,y,z,w) = x \cdot F(x=1) + x' \cdot F(x=0)$ expanded around x

Expand each
cofactor
around y

$$F(x=1) = y \cdot F(x=1, y=1) + \bar{y} F(x=1, y=0)$$

$$F(x=0) = y F(x=0, y=1) + \bar{y} F(x=0, y=0)$$

$$F(x,y,z,w) = x y F(x=1, y=1) + x \bar{y} F(x=1, y=0) + \bar{x} y F(x=0, y=1) + \bar{x} \bar{y} F(x=0, y=0)$$

= expanded around variables x and y

Shannon Cofactors: Multiple Variables

- **BTW, there is notation for these as well**

- Shannon Cofactor with respect to x_i and x_j

- Write $F(x_1, x_2, \dots, x_i=1, \dots, x_j=0, \dots, x_n)$ as $F_{x_i x_j'}$ or $F_{x_i \overline{x_j}}$

- Ditto for any number of variables x_i, x_j, x_k, \dots

- Notice also that order does **not** matter: $(F_x)_y = (F_y)_x = F_{xy}$

- For our example

$$F(x,y,z,w) = xy \cdot F_{xy} + x'y \cdot F_{x'y} + xy' \cdot F_{xy'} + x'y' \cdot F_{x'y'}$$

- Again, **remember**: each of the cofactors is a function, not a number

$F_{xy} = F(x=1, y=1, z, w)$ = a Boolean **function** of z and w



VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

Lecture 2.2

Computational Boolean
Algebra: Boolean Difference



Next Question: Properties of Cofactors

- What **else** can you do with cofactors?

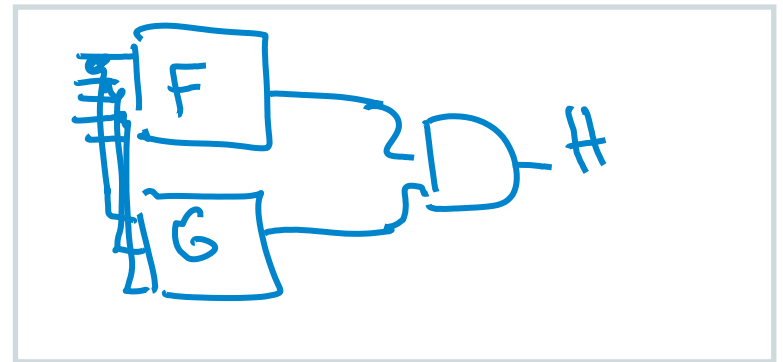
- Suppose you have 2 functions **F(X)** and **G(X)**, where **X=(x1,x2,...xn)**
- Suppose you make a new function **H**, from **F** and **G**, say...

- $H = \overline{F}$

- $H = (F \cdot G)$ ie, $H(X) = F(X) \cdot G(X)$

- $H = (F + G)$ ie, $H(X) = F(X) + G(X)$

- $H = (F \oplus G)$ ie, $H(X) = F(X) \oplus G(X)$



- Interesting question

- Can you tell anything about **H's** cofactors from those of **F, G...**?

$(F \cdot G)_x = \text{what?}$ $(F')_x = \text{what?}$ etc.

Nice Properties of Cofactors

- Cofactors of **F** and **G** tell you *everything* you need to know

- Complements

- $(F')_x = (F_x)'$ $= \overline{(F_x)}$
- In English: *cofactor of complement is complement of cofactor*

- **Binary** boolean operators

- $(F \cdot G)_x = F_x \cdot G_x$ *cofactor of AND is AND of cofactors*
- $(F + G)_x = F_x + G_x$ *cofactor of OR is OR of cofactors*
- $(F \oplus G)_x = F_x \oplus G_x$ *cofactor of EXOR is EXOR of cofactor*



- **Very useful.** can often help in getting cofactors of complex formulas

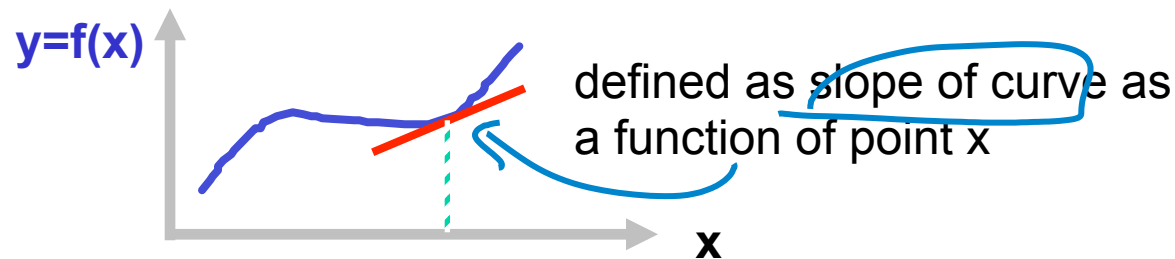
Combinations of Cofactors

- OK, now consider **operations** on cofactors themselves
- Suppose we have $F(X)$, and get F_x and $F_{x'}$
 - $F_x \oplus F_{x'} = ?$
 - $F_x \bullet F_{x'} = ?$
 - $F_x + F_{x'} = ?$
- Turns out these are all useful **new** functions
 - Indeed – they even have **names!**
- Next: let's go look at these interesting, useful new things
 - First up: the **EXOR** of the cofactors

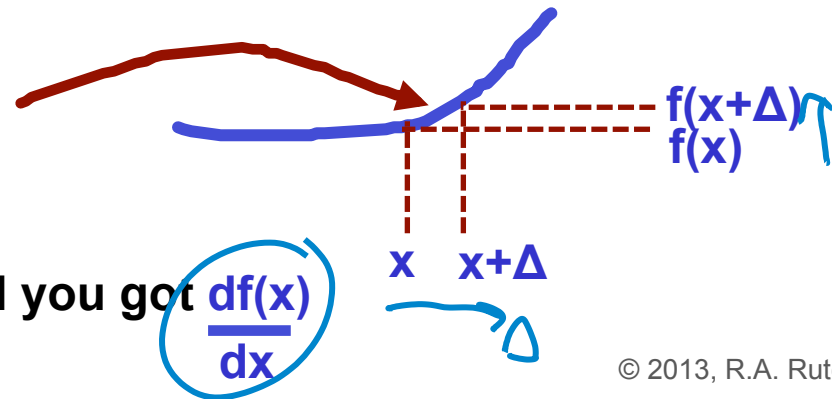


Calculus Revisited: Derivatives

- Remember way back to how you defined derivatives?
 - Suppose you have $y = f(x)$



Considered
$$\frac{f(x + \Delta) - f(x)}{\Delta}$$



Let Δ go to 0 in the limit and you get $\frac{df(x)}{dx}$

Boolean Derivatives

- So, do Boolean functions have “derivatives” ...?

- Actually, **yes**. Trick is how to **define** them...

$$\begin{aligned} f(x) &\oplus f(x) \\ &= a\bar{b} + \bar{a}b \\ &= 1 \text{ iff } a \neq b \end{aligned}$$

- **Basic idea**

- For real-valued $f(x)$, df/dx tell how f changes when x changes
- For **0,1**-valued Boolean function, we cannot change x by small delta
- Can only change $0 \leftrightarrow 1$, but can still ask how f changes with x ...

Boolean $f(x)$:

$$\frac{\partial f}{\partial x} =$$

$$f_x \oplus f_{\bar{x}}$$

Compares value of $f()$ when $x=0$ against when $x=1$;
 $=1$ just if these are *different*

It's Got a Name: Boolean *Difference*

- **Hey, we have seen these pieces before!**
 - $\partial f / \partial x$ = **exor** of the Shannon cofactors with respect to **x**
 - But... for Boolean variables, it's usually written with the ∂ symbol
 - **It also behaves sort of like regular derivatives...**
 - Order of variables (vars) does not matter
 - $\partial f / \partial x \partial y = \partial f / \partial y \partial x$
 - Derivative of exor is exor of derivatives
 - $\partial (f \oplus g) / \partial x = \partial f / \partial x \oplus \partial g / \partial x$
 - If function **f** is actually *constant* (**f=1** or **f=0**, always, for **all** inputs)
 - $\partial f / \partial x = 0$ for any **x**
- like addition*
- like const*

Boolean Difference

- **But some things are just more complex, though...**
 - Derivatives of $(f \bullet g)$ and $(f + g)$ **do not** work the same...

$$\frac{\partial}{\partial x}(f \bullet g) = \left[f \bullet \frac{\partial g}{\partial x} \right] \oplus \left[g \bullet \frac{\partial f}{\partial x} \right] \oplus \left[\frac{\partial f}{\partial x} \bullet \frac{\partial g}{\partial x} \right]$$

$$\frac{\partial}{\partial x}(f + g) = \left[\bar{f} \bullet \frac{\partial g}{\partial x} \right] \oplus \left[\bar{g} \bullet \frac{\partial f}{\partial x} \right] \oplus \left[\frac{\partial f}{\partial x} \bullet \frac{\partial g}{\partial x} \right]$$

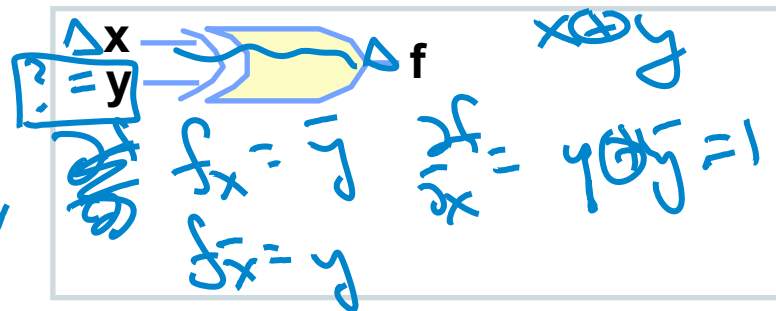
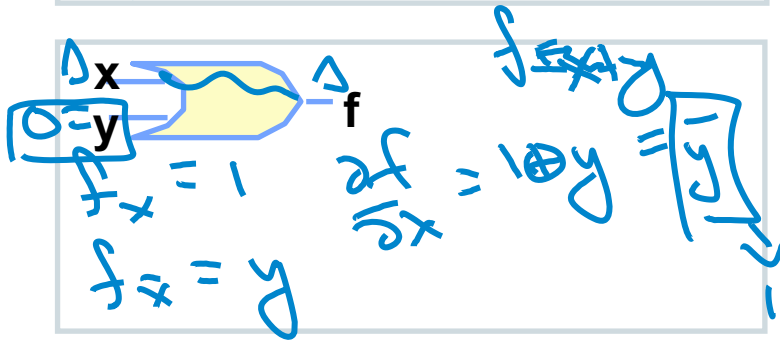
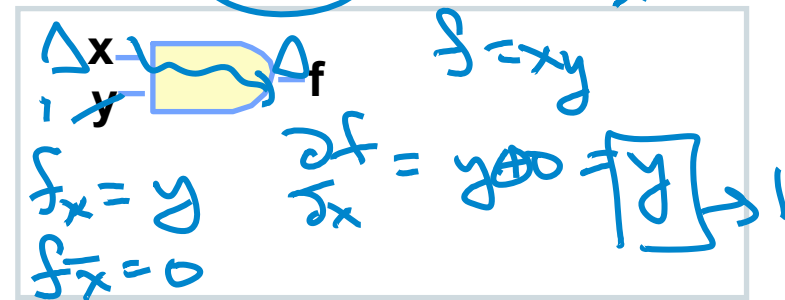
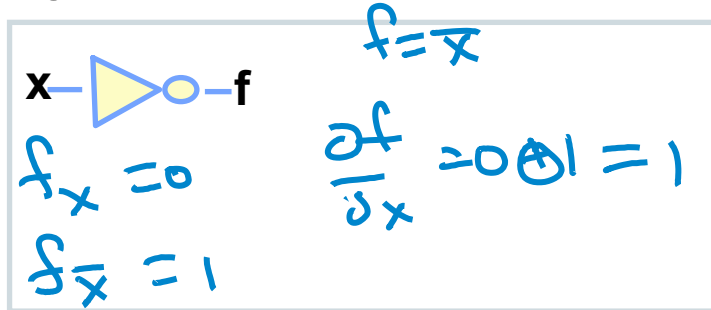
!!

- **Why?**

- Because AND and OR on Boolean values do **not** always behave like ADDITION and MULTIPLICATION on real numbers

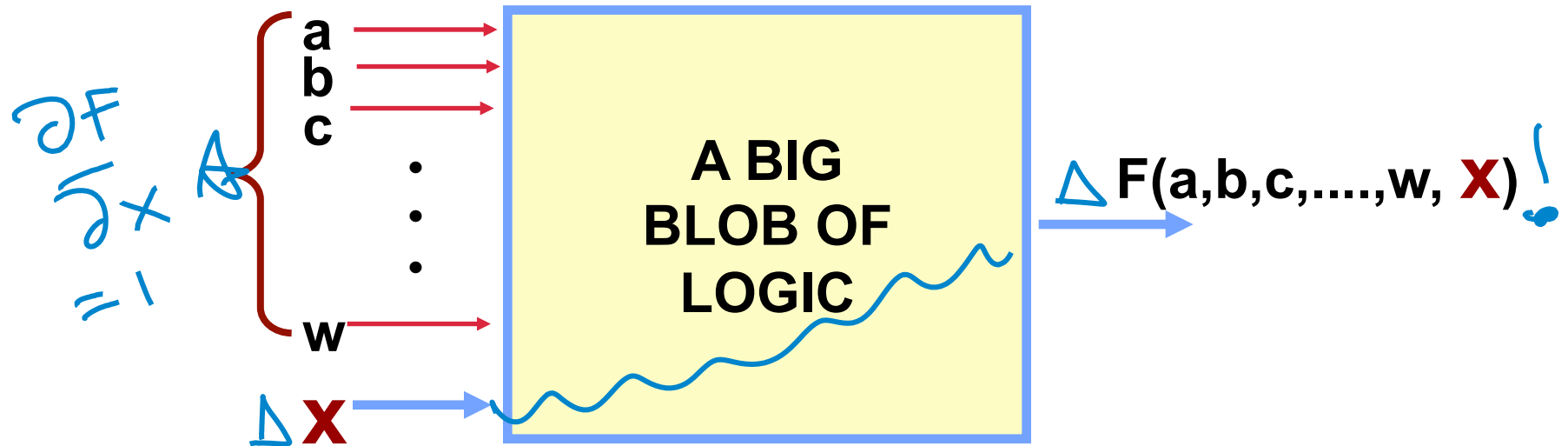
Boolean Difference: Gate-level View

- Try the obvious “simple” examples for $\partial f / \partial x = f_x \oplus f_{\bar{x}}$



Meaning: when $\partial f / \partial x = 1$, then f changes if x changes(!)

Interpreting the Boolean Difference



When $\frac{\partial F}{\partial X}(a,b,c, \dots, w) = 1$, it means that ...

If you apply a pattern of other inputs (not X) that makes $\frac{\partial F}{\partial X} = 1$,
Any change in X will force a change in output $F()$

Boolean Difference: Example



$$\partial Cout / \partial Cin = ?$$

$$\begin{aligned}
 Cout_{cin} &= ab + (a+b)cin = ab \\
 Cout_{\overline{cin}} &= ab \\
 \text{if } a \neq b & \\
 \Delta cin \rightarrow \Delta Cout & \quad ! \leftarrow a \oplus b
 \end{aligned}$$

Handwritten derivation for the Boolean difference:

$$\begin{aligned}
 \partial Cout / \partial cin &= (ab) \oplus ab \\
 &= (\overline{a+b})ab + (a+b)\overline{ab} \\
 &\quad \downarrow \\
 &= a \oplus b
 \end{aligned}$$

Boolean Difference

- **Things to remember about Boolean Difference**
 - Not like the physical interpretation of the ordinary calculus derivative (ie, no “slope of the curve” sort of stuff)...
 - ...but it explains how an input-change can cause output-change for a Boolean $F()$
 - $\partial f / \partial x$ is another Boolean **function**, but it does **not** depend on x
 - It cannot; it is made out of the cofactors with respect to (“wrt”) x
 - ...and they eliminate all the x and x' terms by setting them to constants
- **Surprisingly useful (we will see more, later...)**

