CSE 620: Selective Topics

Introduction to Formal Verification

Master Studies in CSE

Fall 2017

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 10:00am-12:30pm
- Course website

  http://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

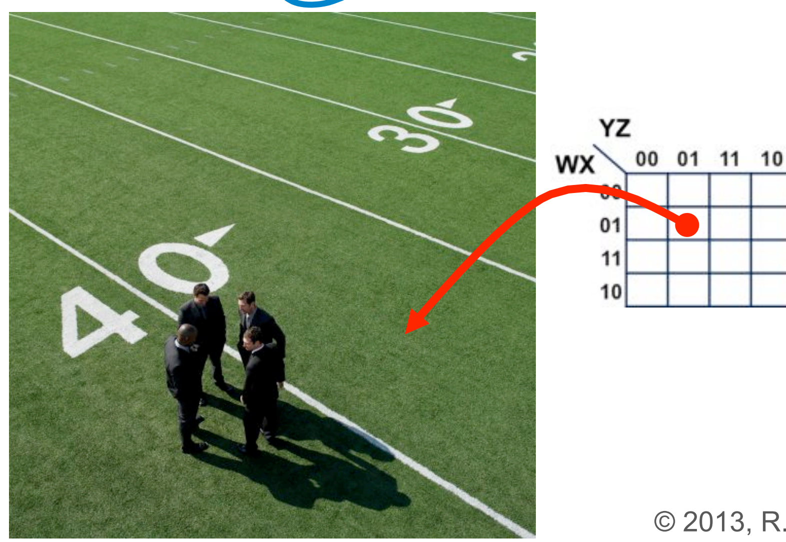
Lecture 2.1

Computational Boolean Algebra: Basics

Rob A. Rutenbar
University of Illinois

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

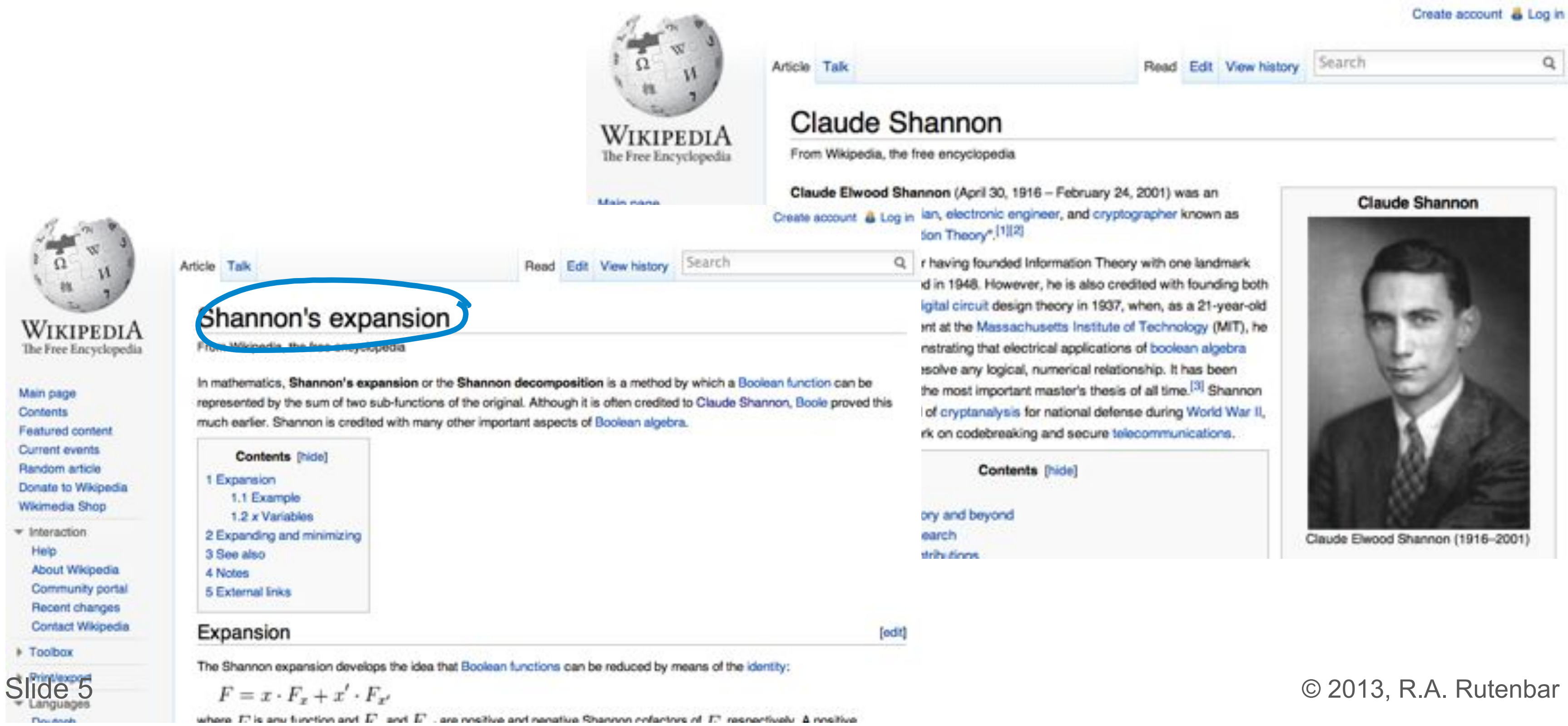
- **Need *algorithmic, computational* strategies for Boolean stuff**
  - Need to be able to think of Boolean objects as **data structures + operators**
- **What will we study?**
  - **Decomposition strategies**
    - Ways of taking apart complex functions into simpler pieces
    - A set of advanced concepts, 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 $\underbrace{\mathbf{1, x, x^2, x^3, x^4, \ldots}}$ as the pieces...
    - ...turns out $\mathbf{exp(x) = 1 + x + x^2/2! + x^3/3! + \ldots}$
  - In Calculus, we tell you the general formula, the **Taylor series expansion**
    - $\mathbf{f(x) = f(0) + f'(0)/1!\, x + f''(0)/2!\, x^2 + f'''(0)/3!\, x^3 + \ldots}$
  - 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***



Shannon Expansion

- **Suppose we have a function F(x1,x2, ..., xn)**
- **Define a new function if we set one of the xi=*constant***
  - Example: **F(x1, x2, ..., xi=1, ..., xn)**
  - Example: **F(x1, x2, ..., xi=0, ..., xn)**
- **Easy to do one by hand**

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

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

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

- **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 **xi**
    - Write **F(x1, x2, ..., xi=1, ...xn)** as: F_{x_i} = positive cofactor
    - Write **F(x1, x2, ..., xi=0, ...xn)** as: $F_{\overline{x_i}}$ - negative cofactor
    - Often write this as just **F(xi=1) F(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(x1, x2, ..., xn)** and pick any **xi** in **F( )'s** inputs **F( )** can be represented as

$$F(x1, x2, \ldots, xi, \ldots\ldots, xn) = xi \bullet \underbrace{F(xi=1)}_{\text{pos}} + xi' \bullet \underbrace{F(xi=0)}_{\text{neg}}$$

- Pretty easy to prove...

Let $x_i = 1$

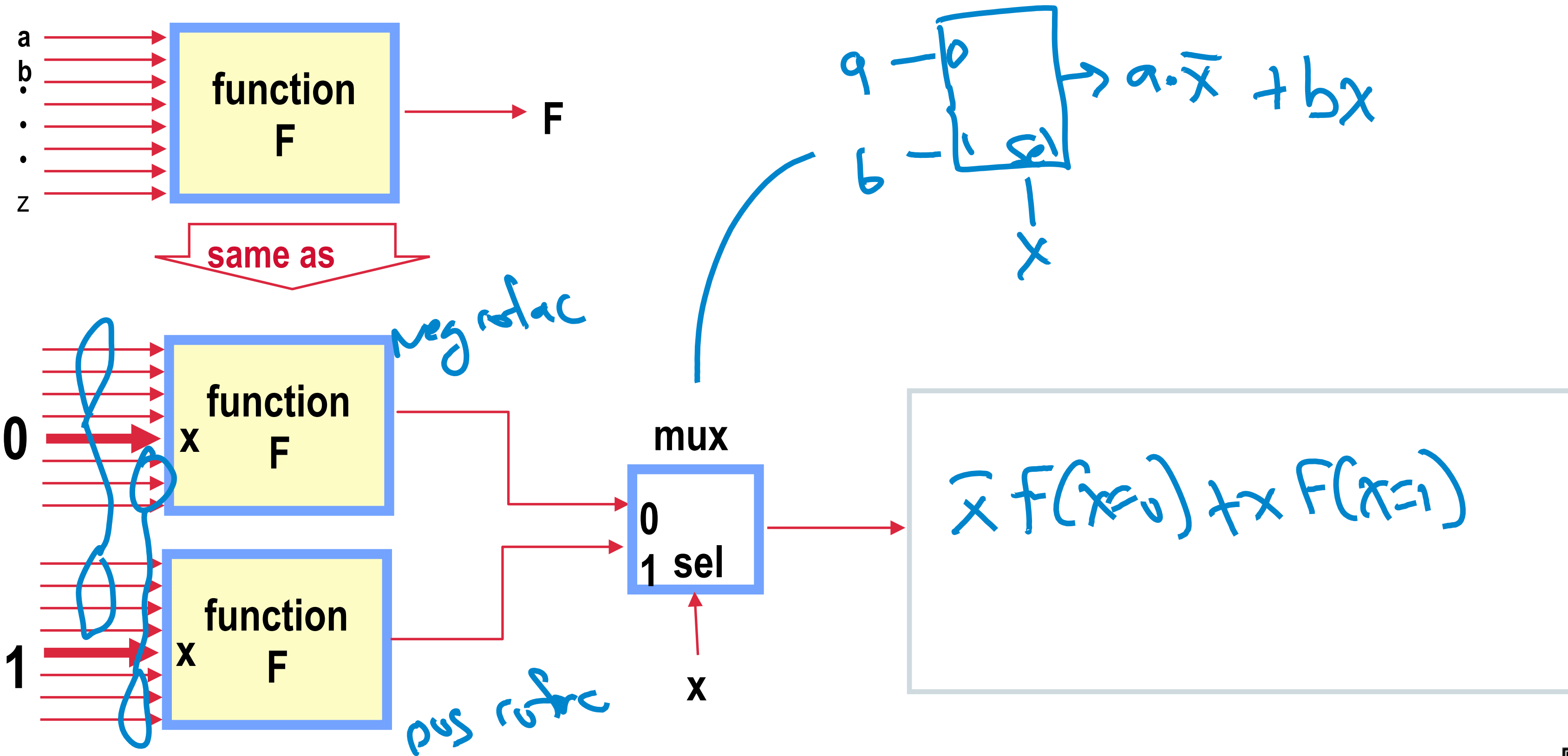
$F(\ldots x_i=1 \ldots) = 1 \bullet F(x_i=1) + \cancel{\overline{1} \bullet F(x_i=0)}$

$= F(x_i=1)$ ✓

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 $F(x,y,z,w) = x \bullet \boxed{F(x=1)} + x' \bullet \boxed{F(x=0)}$ **expanded around x**

Expand each cofactor around y

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

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

$F(x,y,z,w) =$

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

= expanded around variables x and y

Shannon Cofactors: Multiple Variables

- **BTW, there is notation for these as well**
  - Shannon Cofactor with respect to **xi** and **xj**

    $F(x_i=1, x_j=0)$

    - Write **F(x1, x2, ..., xi=1, ..., xj=0, ..., xn)** as $F_{x_i x_j'}$ or $F_{x_i \overline{x_j}}$
    - Ditto for any number of variables **xi, xj, xk**, ...
    - Notice also that order does **not** matter: $(F_x)_y = (F_y)_x = F_{xy}$
  - For our example

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

- 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**

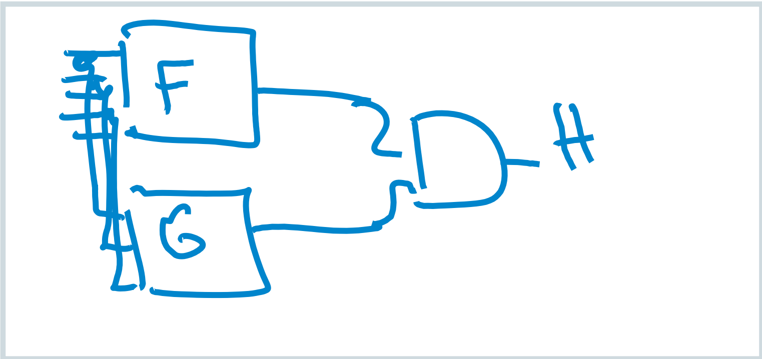
VLSI CAD:
Logic to Layout

Lecture 2.2
Computational Boolean Algebra: Boolean Difference

Rob A. Rutenbar
University of Illinois

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…
    - **$H = \overline{F}$**
    - **$H = (F \bullet G)$ ie, $H(X) = F(X) \bullet G(X)$**
    - **$H = (F + G)$ 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**…?

$(F \bullet G)_x$ = what? $(F\ ')_x$ = 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*

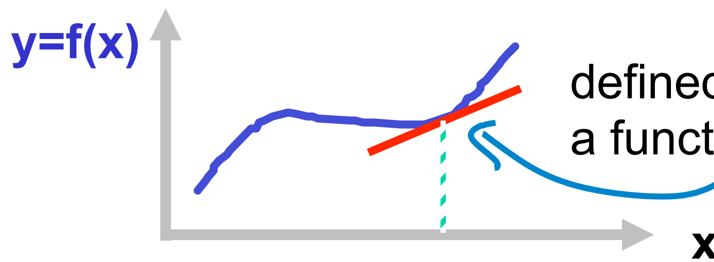
    - **Very useful:** can often help in getting cofactors of complex formulas

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

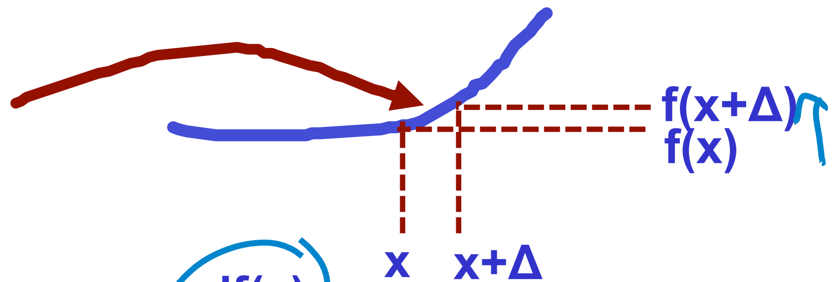
Calculus Revisited: Derivatives

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



defined as slope of curve as a function of point x

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



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

Δ

Boolean Derivatives

- **So, do Boolean functions have "derivatives"…?**
  - Actually, **yes**. Trick is how to **define** them...

Exor $\oplus$ $a \oplus b$

$= a\bar{b} + \bar{a}b$

$=1$ iff $a \neq b$

- **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 ←→ 1**, but can still ask how **f** changes with **x** ...

Boolean f(x):

$\partial f / \partial x$ = $f_x \oplus f_{\bar{x}}$

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
    - $\partial(f \oplus g) / \partial x = \partial f/\partial x \oplus \partial g/\partial x$

like addition

  - If function **f** is actually *constant* (**f=1** or **f=0**, always, for **all** inputs)
    - $\partial f/\partial x = 0$ for any **x**

like const

Boolean Difference

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

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

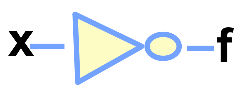
$$\frac{\partial}{\partial x}(f + g) = \left[\bar{f} \bullet \frac{\partial g}{\partial x} \right] \oplus \left[\bar{g} \bullet \frac{\partial f}{\partial x} \right] \oplus \left[\frac{\partial f}{\partial x} \bullet \frac{\partial g}{\partial x} \right]$$

!!

- **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 $\partial f/\partial x$** $= f_x \oplus f_{\bar{x}}$

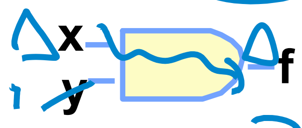


$f = \bar{x}$

$f_x = 0$

$f_{\bar{x}} = 1$

$\frac{\partial f}{\partial x} = 0 \oplus 1 = 1$

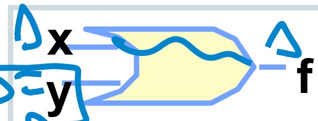


$f = xy$

$f_x = y$

$f_{\bar{x}} = 0$

$\frac{\partial f}{\partial x} = y \oplus 0 = \boxed{y} \rightarrow 1$



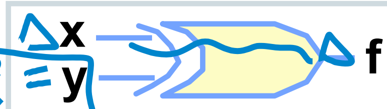
$\boxed{0 = y}$

$f = x + y$

$f_x = 1$

$f_{\bar{x}} = y$

$\frac{\partial f}{\partial x} = 1 \oplus y = \boxed{\bar{y}}$



$\boxed{? = y}$

$x \oplus y$

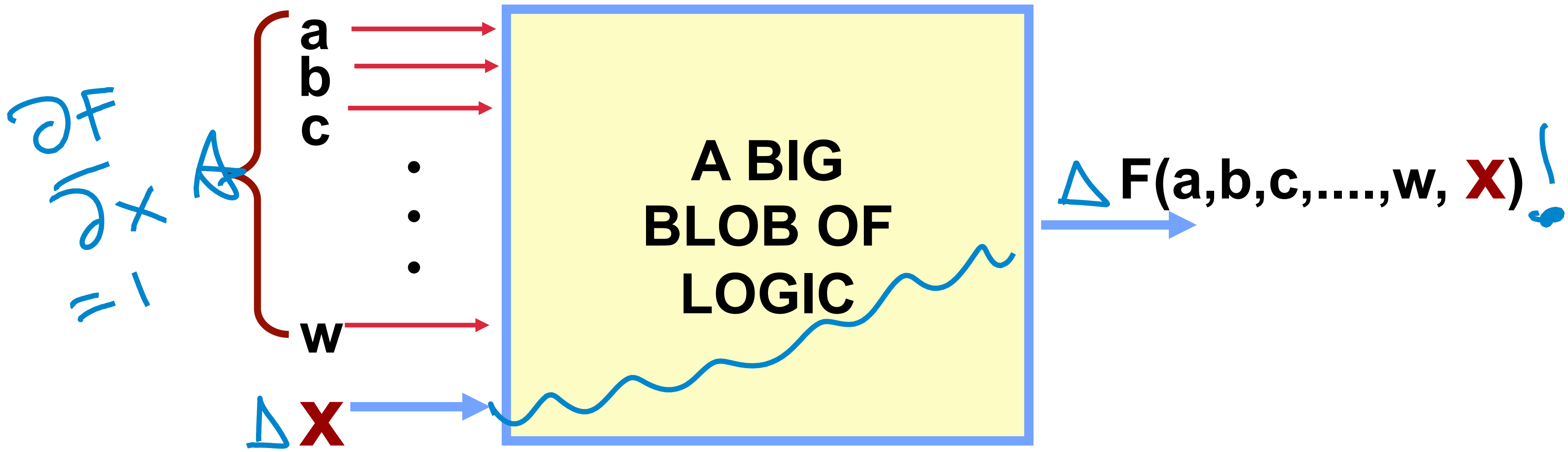
$f_x = \bar{y}$

$f_{\bar{x}} = y$

$\frac{\partial f}{\partial x} = \bar{y} \oplus y = 1$

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

Interpreting the Boolean Difference



When $\partial F / \partial x$ (a,b,c, ..., w) = 1, it means that ...

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

Boolean Difference: Example

s = a ⊕ b ⊕ cin

$\boxed{\text{Cout} = ab + (a + b)\text{Cin}}$

∂Cout / ∂Cin = ?

$Cout_{cin} = ab + (a+b) = a+b$

$Cout_{\overline{cin}} = ab$

$\frac{\partial Cout}{\partial Cin} = (a+b) \oplus ab = \overline{(a+b)}\,ab + (a+b)\,\overline{ab} = a \oplus b$

if $a \neq b$

$\Delta cin \rightarrow \Delta cout$!

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( )
  - **∂f / ∂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…)**