# CENG4120

January 8, 2024

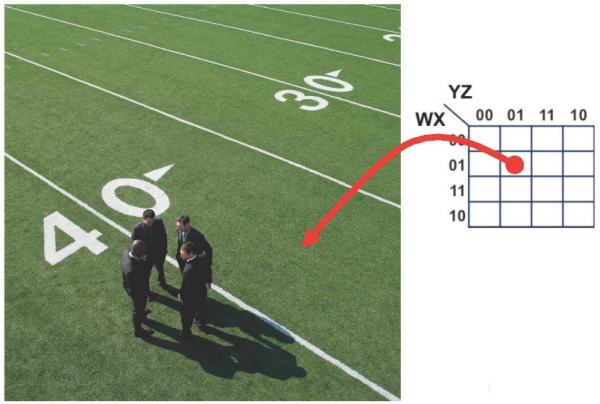
Lecture 2

## *Computational Boolean Algebra Basics*

*(This set of slides sources from R.A. Rutenbar @ UIUC)*

# *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

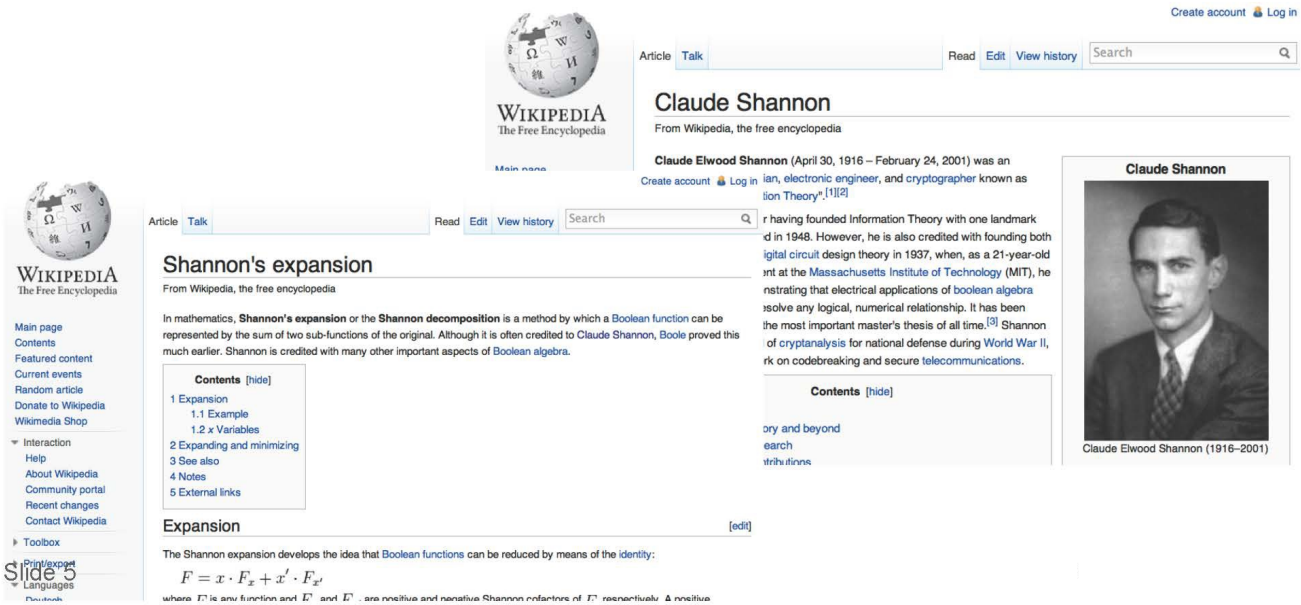
# Advanced Boolean Algebra

- **Useful analogy to calculus...**
  - You can represent complex functions like **exp(x)** using simpler functions
    - If you only get to use **1,x,$x^2$,$x^3$,$x^4$,...** as the pieces...
    - ...turns out **exp(x) = 1 + x + $x^2$/2! + $x^3$/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***



Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
Deutsch

## Shannon's expansion

From Wikipedia, the free encyclopedia

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.

**Contents** [hide]

1 Expansion
1.1 Example
1.2 *x* Variables
2 Expanding and minimizing
3 See also
4 Notes
5 External links

### Expansion [edit]

The Shannon expansion develops the idea that Boolean functions can be reduced by means of the identity:

$$F = x \cdot F_x + x' \cdot F_{x'}$$

where $F$ is any function and $F_x$ and $F_{x'}$ are positive and negative Shannon cofactors of $F$, respectively. A positive

## Claude Shannon

From Wikipedia, the free encyclopedia

**Claude Elwood Shannon** (April 30, 1916 – February 24, 2001) was an ian, electronic engineer, and cryptographer known as ion Theory".[1][2]

r having founded Information Theory with one landmark d in 1948. However, he is also credited with founding both igital circuit design theory in 1937, when, as a 21-year-old nt at the Massachusetts Institute of Technology (MIT), he nstrating that electrical applications of boolean algebra solve any logical, numerical relationship. It has been the most important master's thesis of all time.[3] Shannon of cryptanalysis for national defense during World War II, rk on codebreaking and secure telecommunications.

**Contents** [hide]

ory and beyond
earch
tributions

Claude Shannon



Claude Elwood Shannon (1916–2001)

# 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) =** **F(x,y=0,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:
    - Write **F(x1, x2, ..., xi=0, ...xn)** as:
    - 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?**

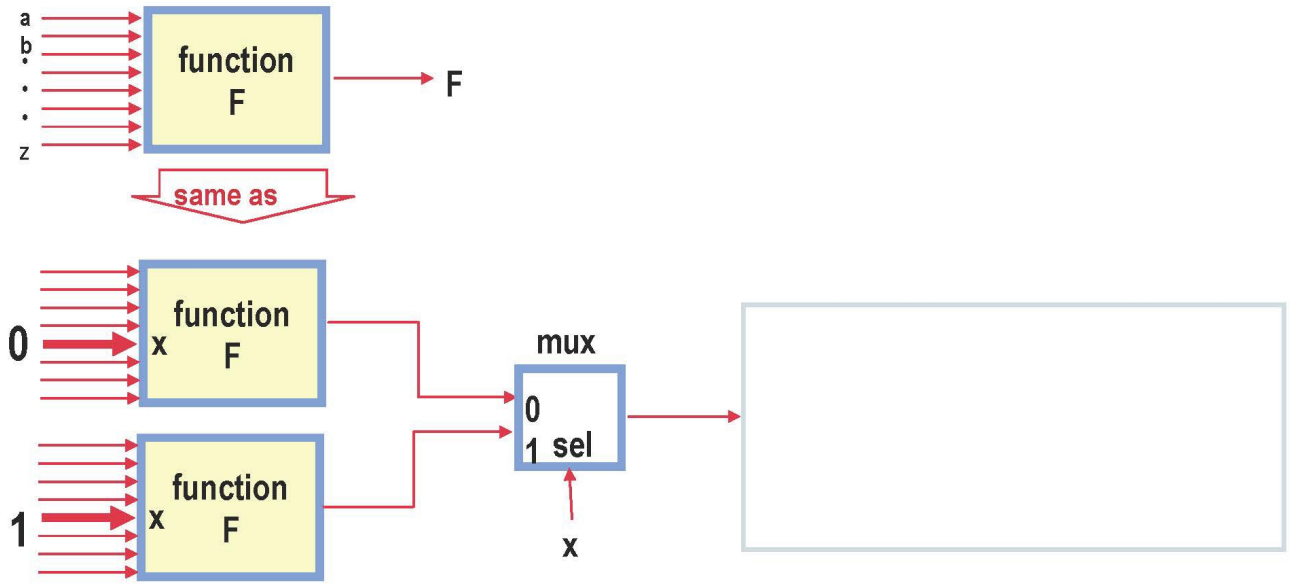
# Shannon Expansion Theorem

- **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

$$F(x1, x2, \ldots, xi, \ldots\ldots, xn) = xi \bullet F(xi=1) + xi' \bullet F(xi=0)$$

pretty easy to prove …

# 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 • F(x=1) + x ' • F(x=0) expanded around x**


F(x,y,z,w) =

= (expand 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**
    - Write **F(x1, x2, ..., xi=1, ..., xj=0, ..., xn)** as $F_{xi\ xj'}$ or $F_{xi\ \overline{xj}}$
    - 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**

# Class Exercise 1

Consider the Boolean function:

$F(a, b, c, d) = abc' + (a' + d)c$

Perform Shannon expansion on variables *a* and *c*.

# 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**…?

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

# Nice Properties of Cofactors

- **Cofactors of F and G tell you *everything* you need to know**
  - **Complements**
    - $(F')_x$ =
    - In English: *cofactor of complement is the 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 $y = f(x)$
- Consider slope defined by $(f(x+\Delta) - f(x)) / \Delta$
- As $\Delta$ goes to 0, we got $df(x)/dx$

# Boolean Derivatives

- So, do Boolean functions have “derivatives”?
  - Actually, yes. The trick is how to define them…

- 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 between 0 and 1, but can still ask how f changes with x…

    df/dx = [ ] = 1 if f() changes with x; and 0 otherwise

# It's Got a Name: Boolean *Difference*

- **Hey, we have seen these pieces before!**
  - **∂f/∂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…(can you prove the following?)**
  - 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***

# Boolean Difference

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

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

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

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

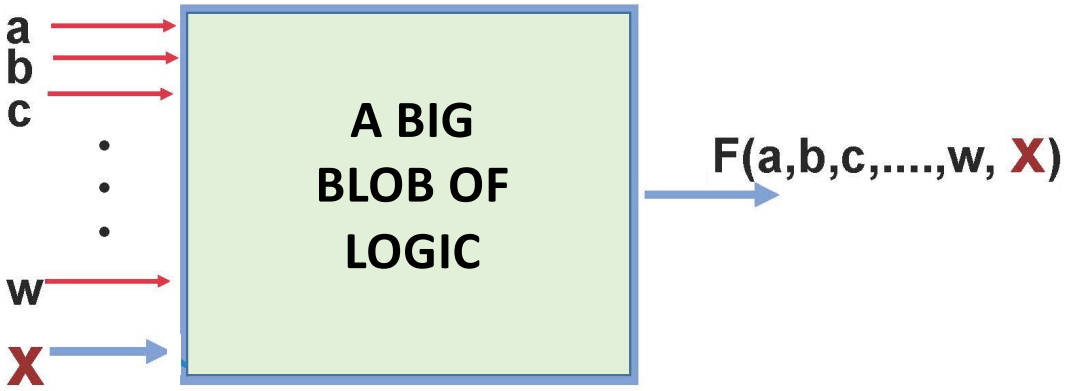
# Boolean Difference: Gate-level View

Consider df/dx when (a) f = x', (b) f = x+y, (c) f = xy, and (d) f = x ⊕ y

Class Exercise 2

Meaning: when df/dx = 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


$s = a \oplus b \oplus cin$

$Cout = ab + (a + b)\,Cin$

$\partial Cout / \partial Cin = ?$

Class Exercise 3

# 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…)**

# Computational Boolean Algebra, Cont…

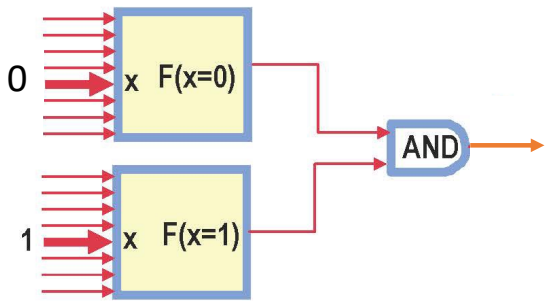
- **What you know**
  - Shannon expansion lets you decompose a Boolean function
  - Combinations of cofactors do interesting things, e.g., the Boolean difference

- **What you don't know**
  - Other combinations of cofactors that do useful things
    - The big ones: **Quantification operators**
    - **Applications:** Being able to do something impressive

# AND: Fx • Fx' is *Universal Quantification*

- **Have F(x1, x2, …, xi, … xn)**
- **AND cofactors: $F_{xi} \cdot F_{xi'}$**
  - Name: **Universal Quantification** of function **F** with respect to (wrt) variable **xi**

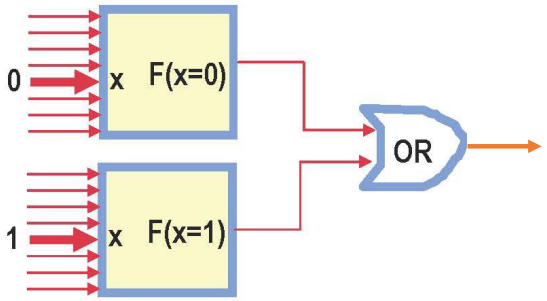
**(∀xi F) [x1,x2, …, xi-1, xi+1, … xn]**




- **"(∀xi F)" is a new function**
  - Yes, the "∀" sign is the "for all" symbol from logic (predicate calculus)
  - And, it does not depend on **xi**…

# OR: Fx + Fx' is *Existential Quantification*

- **Have F(x1, x2, …, xi, … xn)**
- **OR the cofactors:** $F_{xi} + F_{xi'}$
  - Name: **Existential Quantification** of function **F** wrt variable **xi**
  - **(∃xi F) [x1, x2, …, xi-1, xi+1, … xn]**

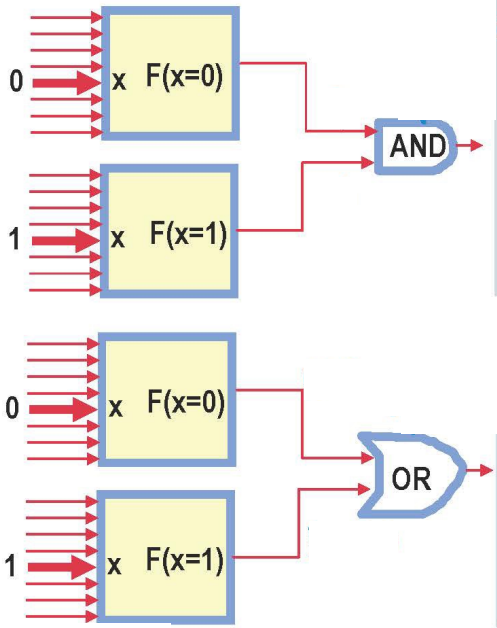



- **"(∃xi F)" is a new function**
  - "∃" sign is "there exists" from logic; and function also does not depend on **xi**

**Note**, like anything involving cofactors, both these new functions **do not** depend on **xi**

So: **(∀xi F)** and **(∃xi F)** both omit **xi**

# Quantification Notation Makes Sense…




**(∀ x F) ( all original vars but x) == 1 *when?***

**(∃ x F) ( all original vars but x) == 1 *when?***

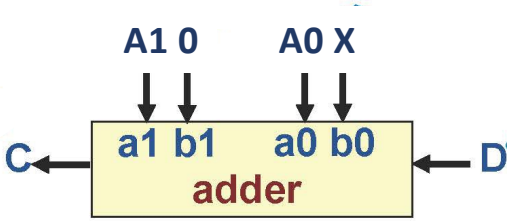
# Extends to More Variables in Obvious Way

- **Additional properties**
  - Like Boolean difference, can do with respect to **more** than 1 var
  - Suppose we have **F(x,y,z,w)**
  - Example: $(\forall xy\ F)[z,w] = (\forall x\ (\forall y\ F)) = Fxy \cdot Fx'y \cdot Fxy' \cdot Fx'y'$
  - Example: $(\exists xy\ F)[z,w] = (\exists x\ (\exists y\ F)) = Fxy + Fx'y + Fxy' + Fx'y'$

- **Remember!**
  - $(\forall x\ F)$, $(\exists x\ F)$, and $\partial F / \partial x$ are all **functions**...
  - ..but they are functions of all the vars **except x**
  - We got rid of variable x and made 3 **new** functions

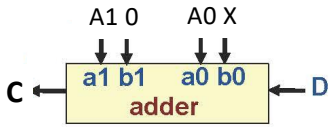
# Quantification Example

- **Consider this circuit, it adds X=0 or X=1 to a 2-bit number A1A0**
  - It's just a 2-bit adder, but instead of **B1B0** for the second operand, it is just **0X**
  - It produces a carry out called **C** and also has a carry in called **D**




  - What is **(∀A1,A0 C)[X,D]…?**
    - A function of only **X,D**. Makes a **1** for values of **X,D** that make carry **C=1** for ***all values*** of operand input **A1A0**, i.e., makes a carry **C=1** for all values of **A1A0**
  - What is **(∃A1,A0 C)[X,D]…?**
    - A function of just **X,D**. Makes a **1** for values of **X,D** that make carry **C=1**, for ***some value*** of **A1A0**, i.e., ***there exists*** some **A1A0** that, for this **X**,**D** makes **C=1**

# Quantification Example




Do the math. Need all 4 cofactors:

$C = A1A0X + A1(A0+X)D$

$C_{A1A0}$ $C_{A1'A0}$ $C_{A1A0'}$ $C_{A1'A0'}$

- **Compute ($\forall$ A1,A0 C)[X,D]**
  - $C_{A1A0} \bullet C_{A1'A0} \bullet C_{A1A0'} \bullet C_{A1'A0'}$
- **In words: No** values of **X,D** that make **C=1** independent of **A1,A0**

- **Compute ($\exists$ A1,A0 C)[X,D]**
  - $C_{A1A0} + C_{A1'A0} + C_{A1A0'} + C_{A1'A0'}$
- **In words: Yes**, if at least one of **X,D = 1 $\rightarrow$ C=1 for some A1,A0**