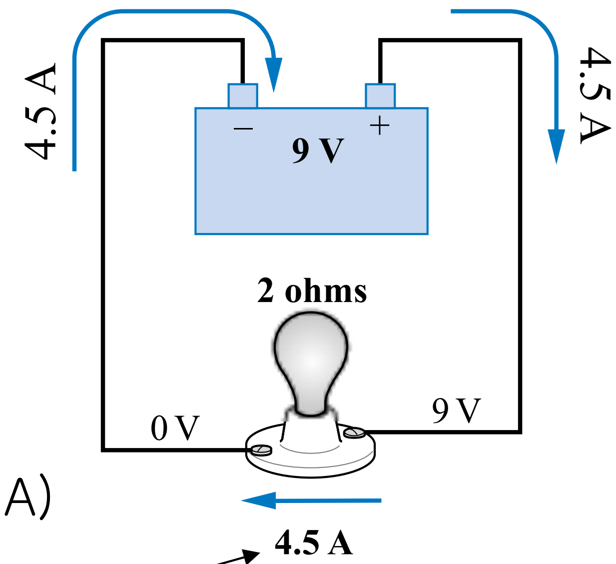
Digital Circuit, Logic, Boolean Algebra, and Truth Table

교재 2장 1절 ~ 5절

Switches

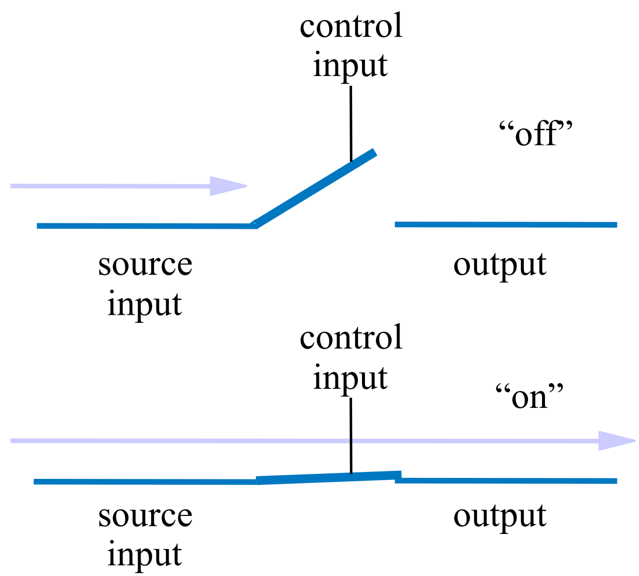
- Electronic switches are the basis of binary digital circuits
  - Electrical terminology
    - *Voltage*: Difference in electric potential between two points (volts, V)
      - Analogous to water pressure
    - *Resistance*: Tendency of wire to resist current flow (ohms, Ω)
      - Analogous to water pipe diameter
    - *Current*: Flow of charged particles (amps, A)
      - Analogous to water flow
    - V = I * R (Ohm's Law)
      - 9 V = I * 2 ohms
      - I = 4.5 A



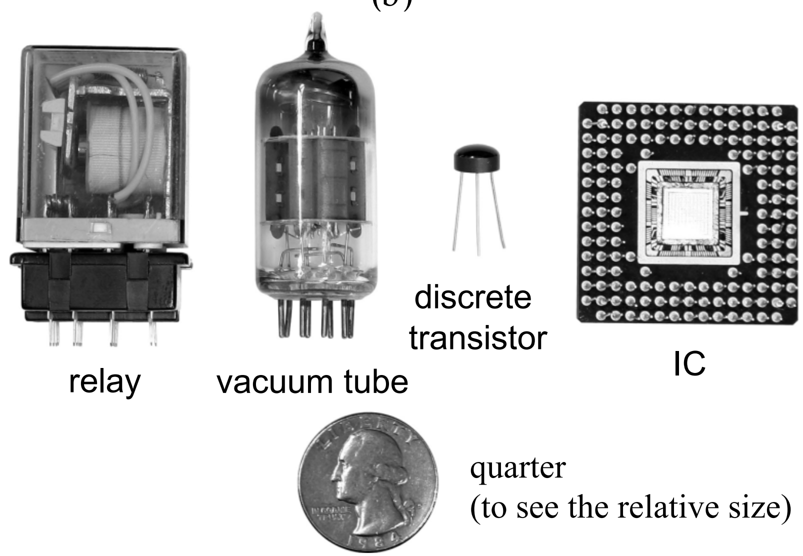
If a 9V potential difference is applied across a 2 ohm resistor, then 4.5 A of current will flow.

Switches

- A switch has three parts
  - Source input, and output
    - Current tries to flow from source input to output
  - Control input
    - Voltage that controls whether that current can flow
- The amazing shrinking switch
  - 1930s: Relays
  - 1940s: Vacuum tubes
  - 1950s: Discrete transistor
  - 1960s: Integrated circuits (ICs)
    - Initially just a few transistors on IC
    - Then tens, hundreds, thousands...

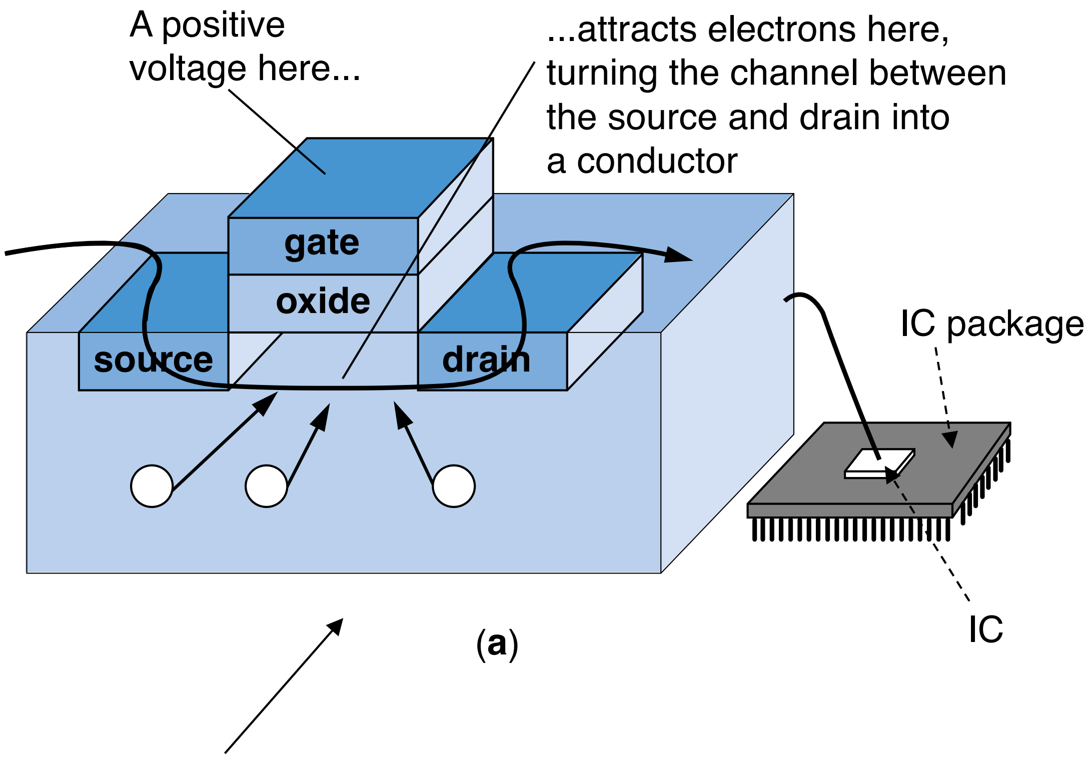


(b)

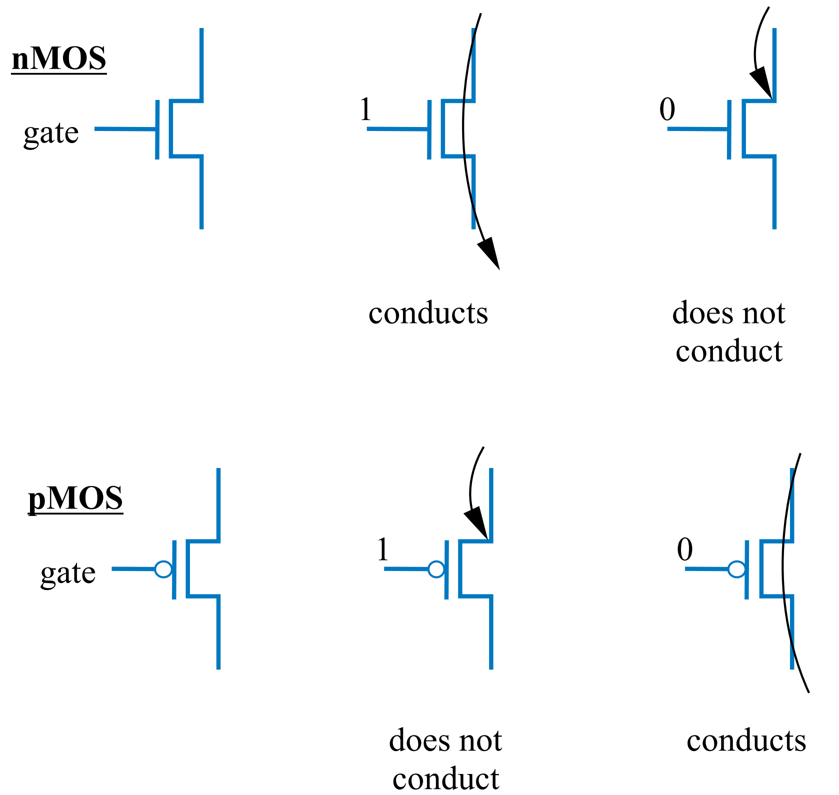


The CMOS Transistor

- CMOS transistor
  - Basic switch in modern ICs

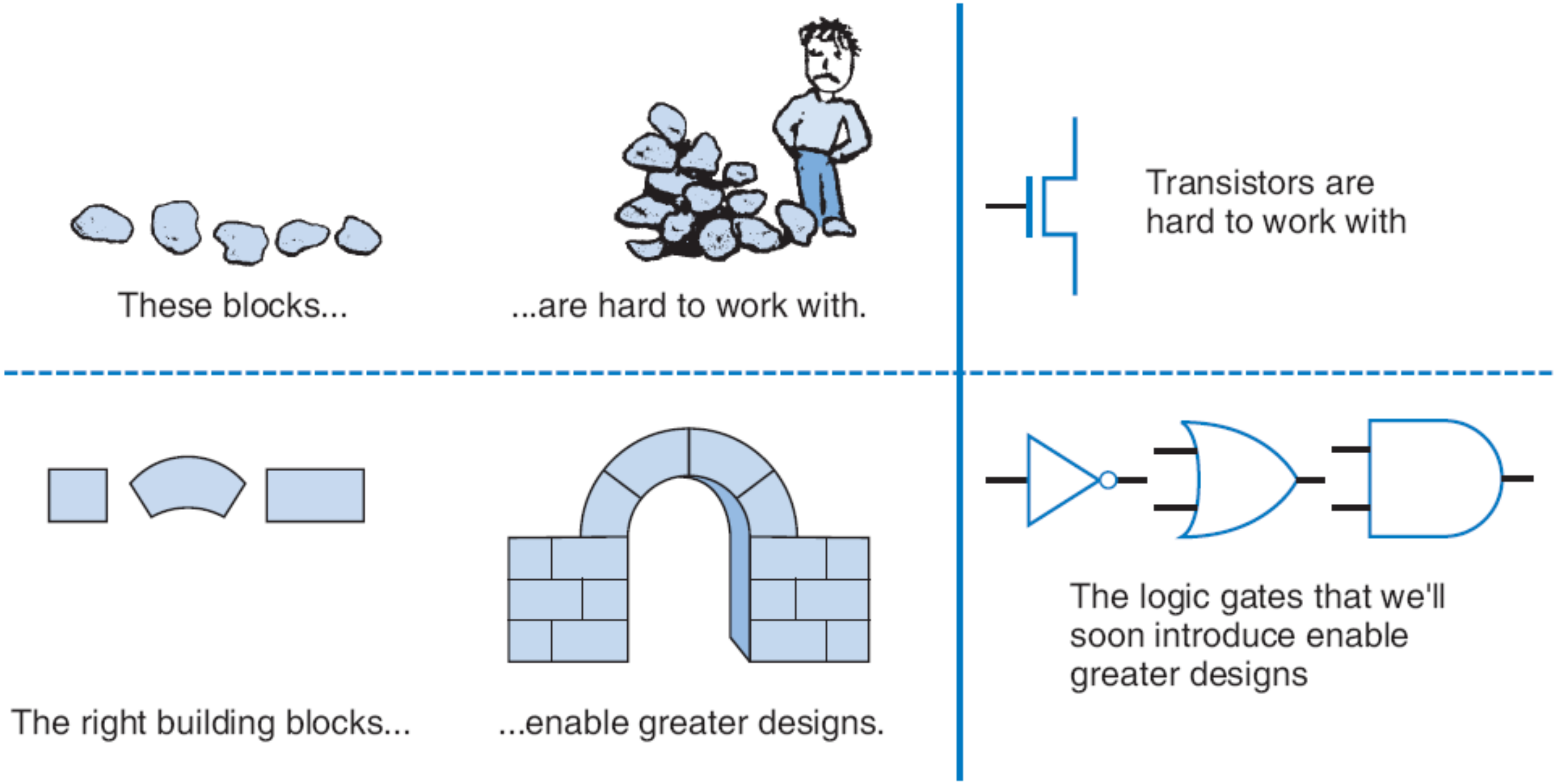


Silicon -- not quite a conductor or insulator:
Semiconductor



Boolean Logic Gates
Building Blocks for Digital Circuits

(Because Switches are Hard to Work With)



- "Logic gates" are better digital circuit building blocks than switches (transistors)
  - Why?...

Boolean Algebra and its Relation to Digital Circuits

- To understand the benefits of "logic gates" vs. switches, we should first understand Boolean algebra
- "Traditional" algebra
  - Variables represent real numbers (x, y)
  - Operators operate on variables, return real numbers (2.5*x + y – 3)
- *Boolean Algebra*
  - Variables represent 0 or 1 only
  - Operators return 0 or 1 only
  - Basic operators
    - AND: *a AND b* returns 1 only when both a=1 and b=1
    - OR: *a OR b* returns 1 if either (or both) a=1 or b=1
    - NOT: *NOT a* returns the opposite of a (1 if a=0, 0 if a=1)

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0

Boolean Algebra and its Relation to Digital Circuits

- Developed mid-1800's by George Boole to formalize human thought
  - Ex: "I'll go to lunch if Mary goes OR John goes, AND Sally does not go."
    - Let F represent my going to lunch (1 means I go, 0 I don't go)
    - Likewise, m for Mary going, j for John, and s for Sally
    - Then **F = (m OR j) AND NOT(s)**
  - Nice features
    - Formally evaluate
      - m=1, j=0, s=1 --> F = (1 OR 0) AND NOT(1) = 1 AND 0 = <u>0</u>
    - Formally transform
      - F = (m and NOT(s)) OR (j and NOT(s))
        - » Looks different, but same function
        - » We'll show transformation techniques soon
    - Formally prove
      - Prove that if Sally goes to lunch (s=1), then I don't go (F=0)
      - F = (m OR j) AND NOT(1) = (m OR j) AND 0 = 0

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	1
1	0

Relating Boolean Algebra to Digital Design

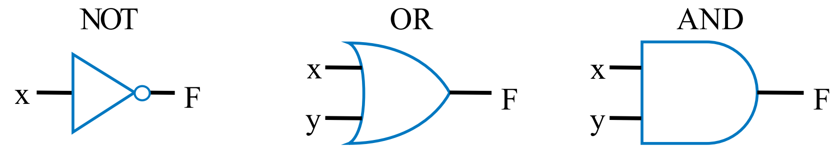
Boolean algebra (mid-1800s) — *Boole's intent: formalize human thought*

Switches (1930s) — *For telephone switching and other electronic uses*

Shannon (1938) — *Showed application of Boolean algebra to design of switch-based circuits*

Digital design

Symbol



Truth table

NOT

x	F
0	1
1	0

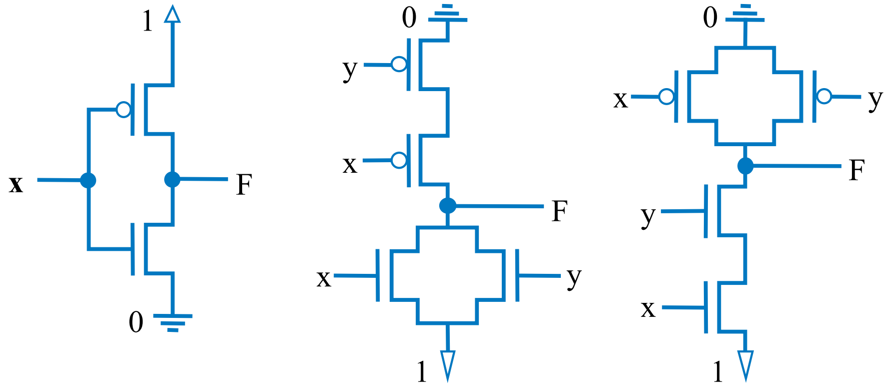
OR

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

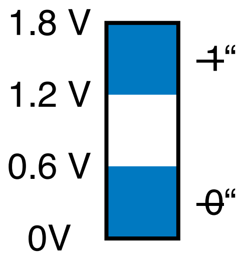
AND

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

Transistor circuit



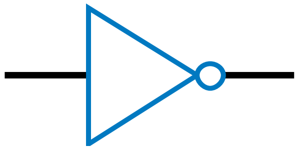
- Implement Boolean operators using transistors
  - Call those implementations *logic gates*.
  - ***<u>Lets us build circuits by doing math</u>*** – – powerful concept



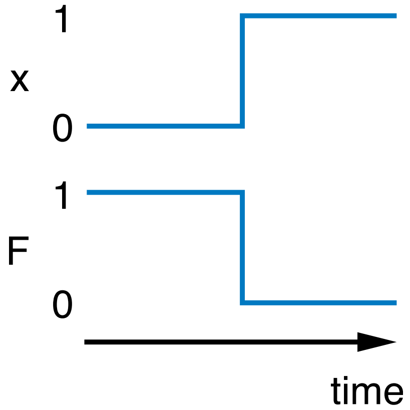
1 and 0 each actually corresponds to a voltage range

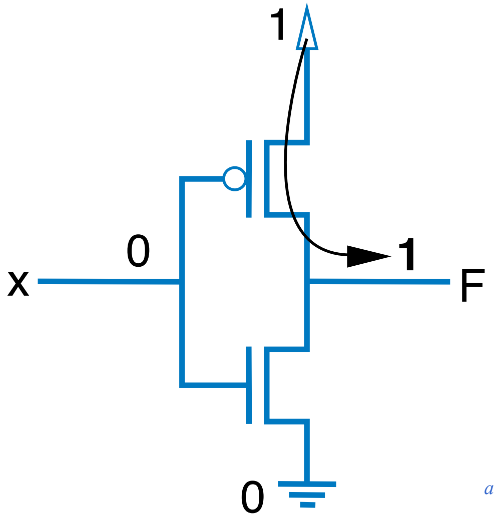
Next slides show how these circuits work. <u>Note</u>: The above OR/AND implementations are inefficient; we'll show why, and show better ones, later.

NOT gate



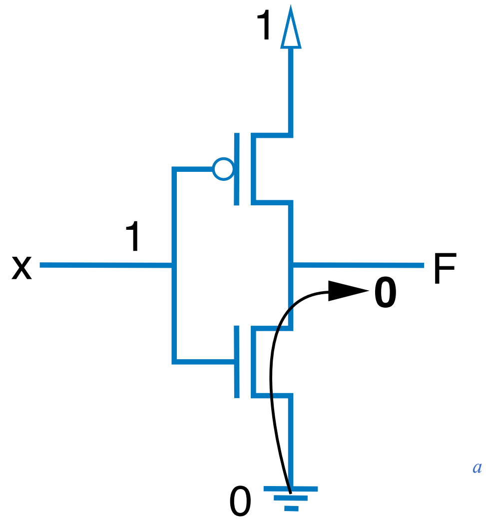
x	F
0	1
1	0





(a)

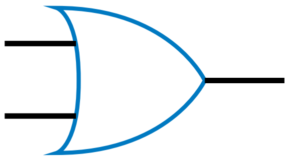
When the input is 0



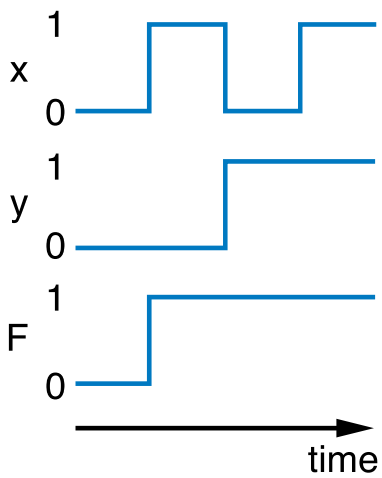
(b)

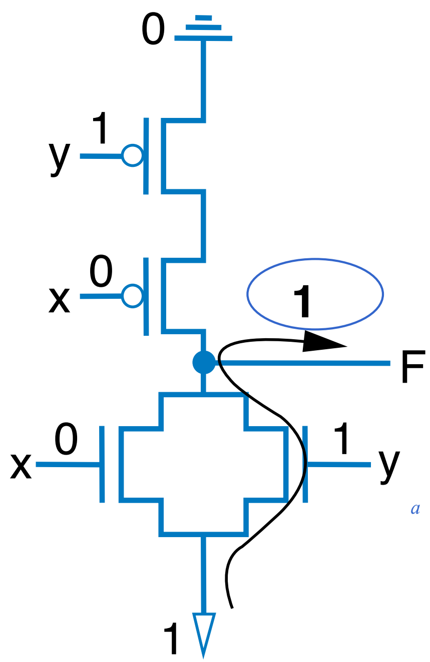
When the input is 1

OR gate

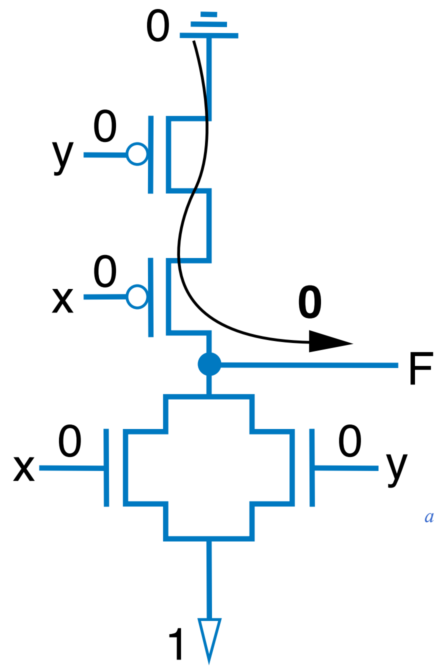


x	y	F
0	0	0
0	1	1
1	0	1
1	1	1



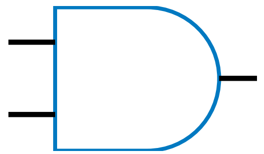


When an input is 1

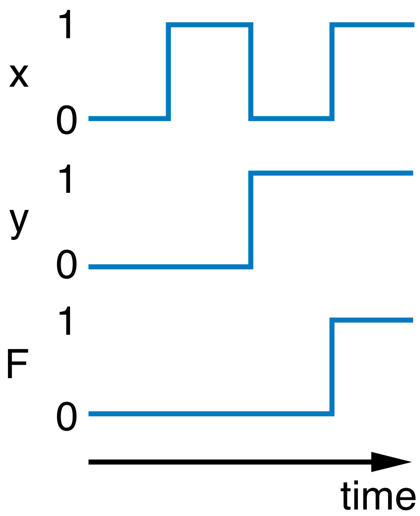


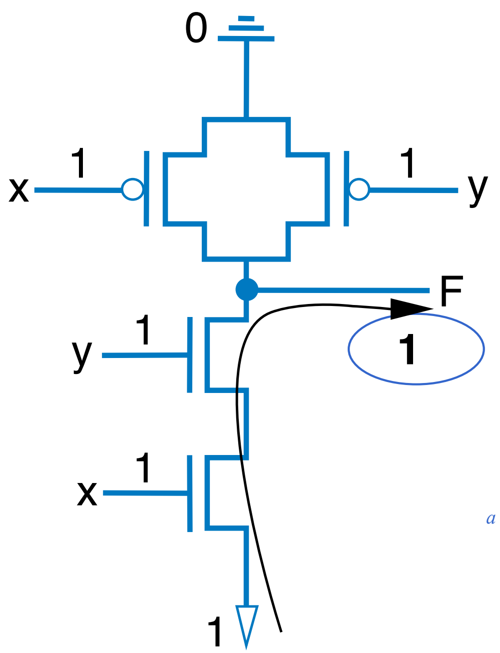
When both inputs are 0

AND gate



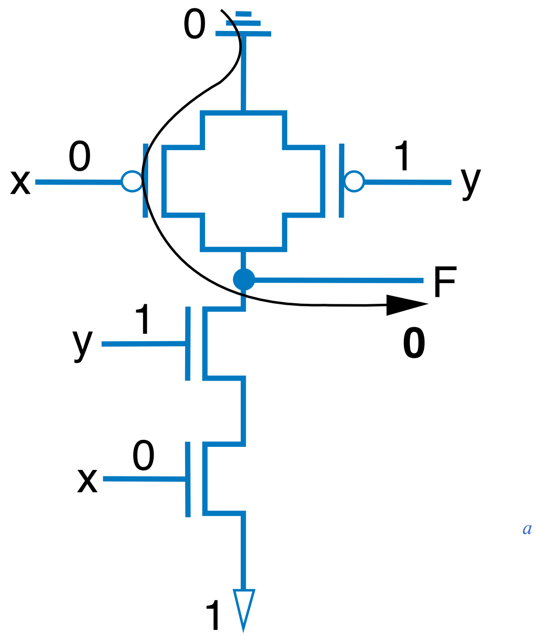
x	y	F
0	0	0
0	1	0
1	0	0
1	1	1





(a)

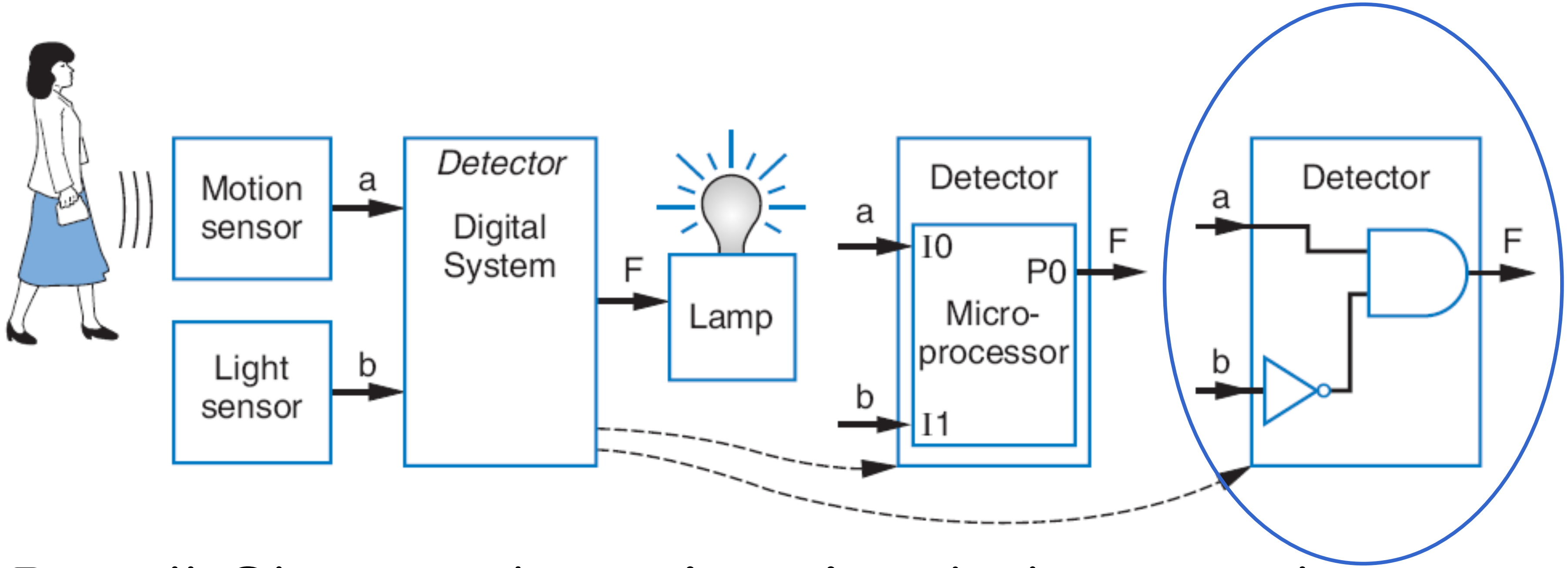
When both inputs are 1



(b)

When an input is 0

Building Circuits Using Gates



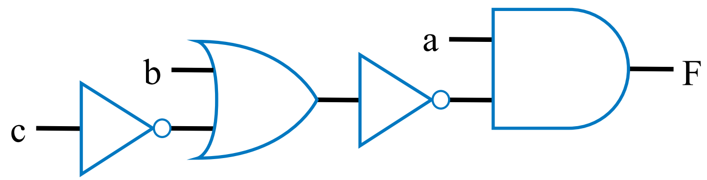
- Recall Chapter 1 motion-in-dark example
  - Turn on lamp (F=1) when motion sensed (a=1) and no light (b=0)
  - F = a AND NOT(b)
  - Build using logic gates, AND and NOT, as shown
  - We just built our first digital circuit!

Example: Converting a Boolean Equation to a Circuit of Logic Gates

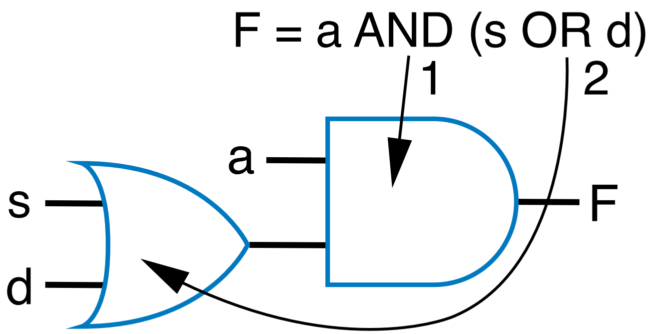
Start from the output, work back towards the inputs

- Q: Convert the following equation to logic gates:
  F = a AND NOT(b OR NOT(c))

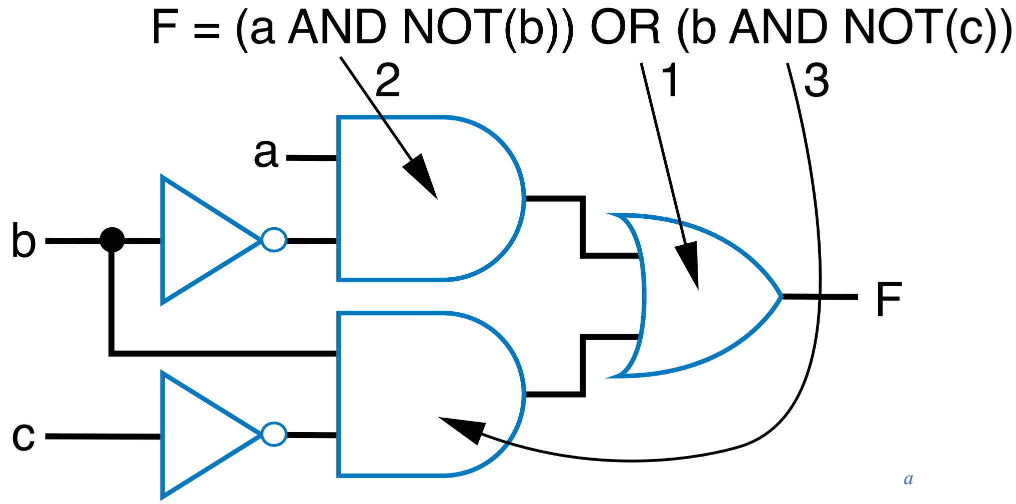
a



More examples



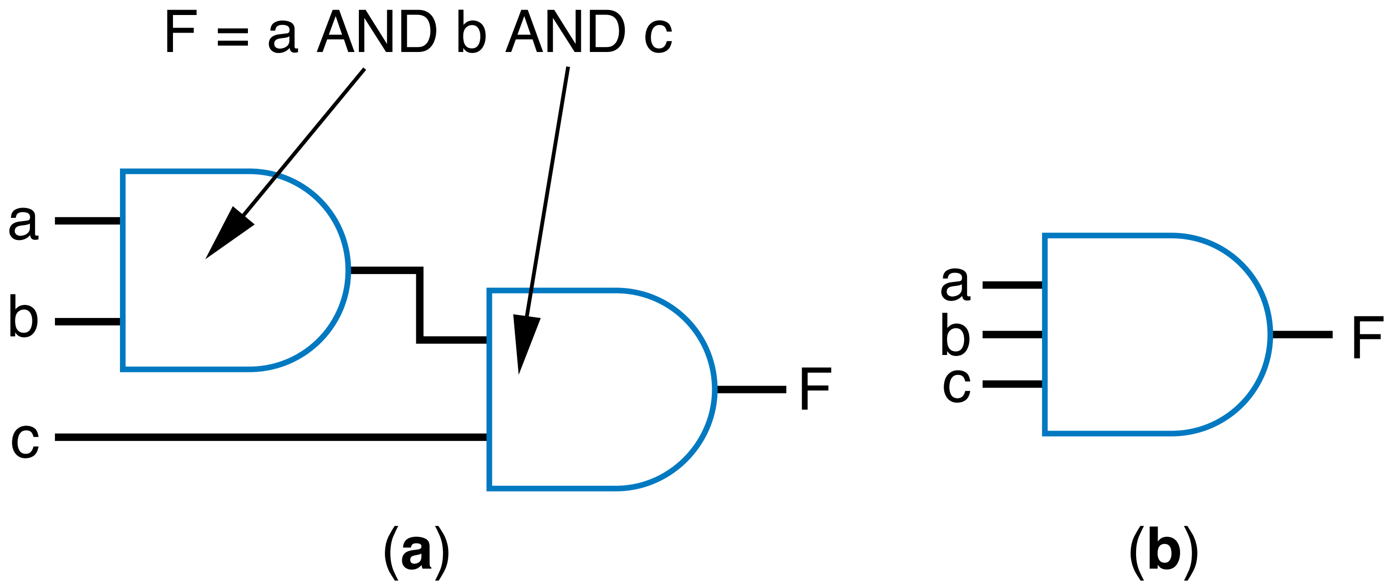
(a)



(b)

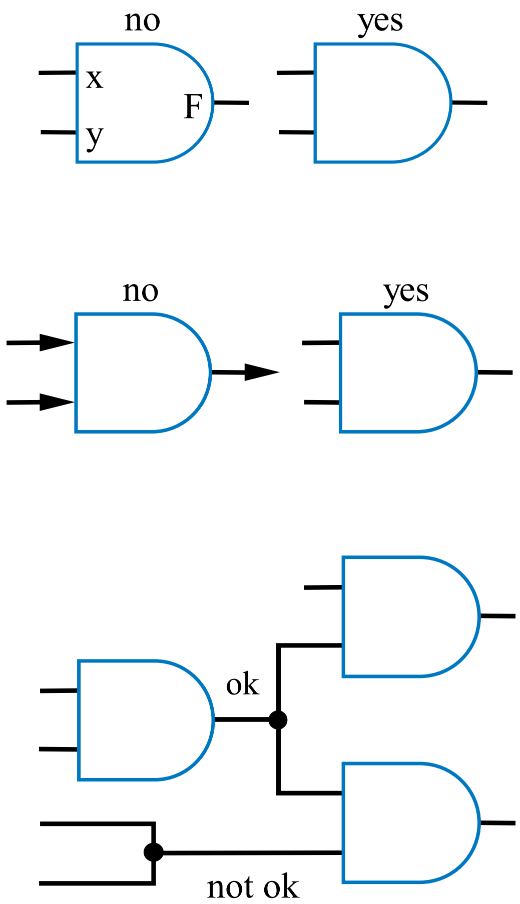
Start from the output, work back towards the inputs

Using gates with more than 2 inputs



Can think of as AND(a,b,c)

Some Gate-Based Circuit Drawing Conventions

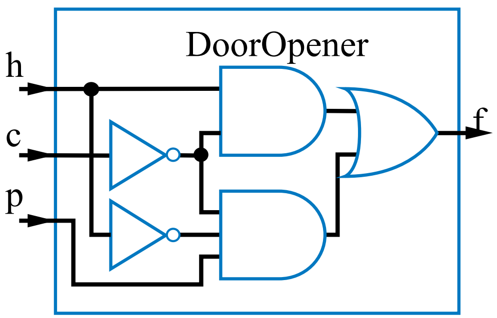


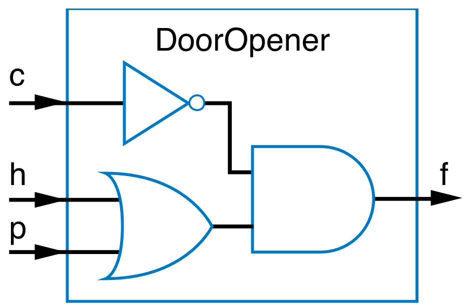
Example that Applies Boolean Algebra Properties

- Want automatic door opener circuit (e.g., for grocery store)
  - Output: f=1 opens door
  - Inputs:
    - p=1: person detected
    - h=1: switch forcing hold open
    - c=1: key forcing closed
  - Want open door when
    - h=1 and c=0, or
    - h=0 and p=1 and c=0
  - Equation: f = hc' + h'pc'

- Can the circuit be simplified?

$f = hc' + h'pc'$	
$f = c'h + c'h'p$	(by the commutative property)
$f = c'(h + h'p)$	(by the first distrib. property)
$f = c'((h+h')*(h+p))$	(2nd distrib. prop.; tricky one)
$f = c'((1)*(h + p))$	(by the complement property)
$f = c'(h+p)$	(by the identity property)





Simplified circuit

Simplification of circuits is covered in Sec. 2.11 / Sec 6.2.

Boolean Algebra

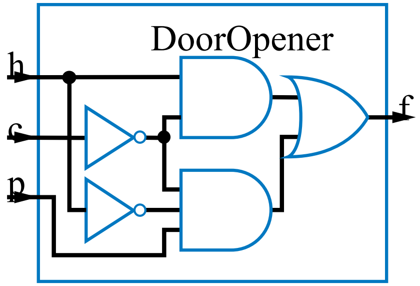
- By defining logic gates based on Boolean algebra, we can *use algebraic methods to manipulate circuits*
- Notation: Writing a AND b, a OR b, NOT(a) is cumbersome
  - Use symbols: a * b (or just ab), a + b, and a’
    - Original: w = (p AND NOT(s) AND k) OR t
    - New: w = ps’k + t
      - Spoken as “w equals p and s prime and k, or t”
      - Or just “w equals p s prime k, or t”
      - s’ known as “complement of s”
    - While symbols come from regular algebra, ***don’t*** say “times” or “plus”
      - "product" and "sum" are OK and commonly used

Boolean algebra precedence, highest precedence first.

Symbol	Name	Description
( )	Parentheses	Evaluate expressions nested in parentheses first
’	NOT	Evaluate from left to right
*	AND	Evaluate from left to right
+	OR	Evaluate from left to right

Example that Applies Boolean Algebra Properties

- Found inexpensive chip that computes:
  - $f = c'hp + c'hp' + c'h'p$
  - Can we use it for the door opener?
    - Is it the same as $f = hc' + h'pc'$?
- Apply Boolean algebra:



- Commutative
  - a + b = b + a
  - a * b = b * a
- Distributive
  - a * (b + c) = a * b + a * c
  - a + (b * c) = (a + b) * (a + c)
- Associative
  - (a + b) + c = a + (b + c)
  - (a * b) * c = a * (b * c)
- Identity
  - 0 + a = a + 0 = a
  - 1 * a = a * 1 = a
- Complement
  - a + a' = 1
  - a * a' = 0

$f = c'hp + c'hp' + c'h'p$

$f = c'h(p + p') + c'h'p$ (by the distributive property)

$f = c'h(1) + c'h'p$ (by the complement property)

$f = c'h + c'h'p$ (by the identity property)

$f = hc' + h'pc'$ (by the commutative property)

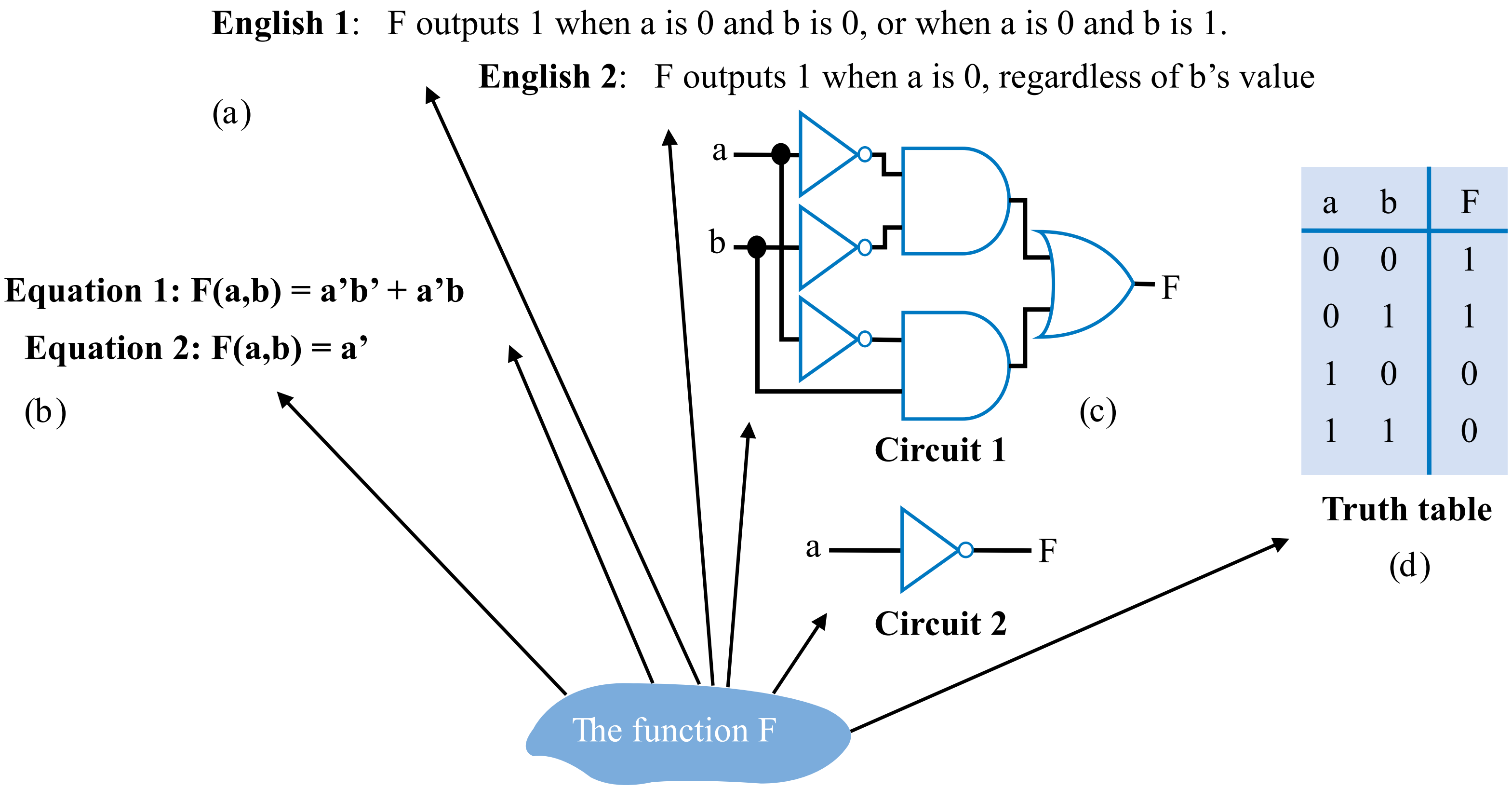
Same! Yes, we can use it.

a

Boolean Algebra: Additional Properties

- Null elements
  - a + 1 = 1
  - a * 0 = 0
- Idempotent Law
  - a + a = a
  - a * a = a
- Involution Law
  - (a')' = a
- DeMorgan's Law
  - (a + b)' = a'b'
  - (ab)' = a' + b'
  - *Very useful!*
- To prove, just evaluate all possibilities

Representations of Boolean Functions



a	b	F
0	0	1
0	1	1
1	0	0
1	1	0

- A function can be represented in different ways
  - Above shows seven representations of the same functions F(a,b), using four different methods: English, Equation, Circuit, and Truth Table

Truth Table Representation of Boolean Functions

- Define value of F for each possible combination of input values
  - 2-input function: 4 rows
  - 3-input function: 8 rows
  - 4-input function: 16 rows
- Q: Use truth table to define function F(a,b,c) that is 1 when abc is 5 or greater in binary

a	b	F
0	0	
0	1	
1	0	
1	1	

(a)

a	b	c	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

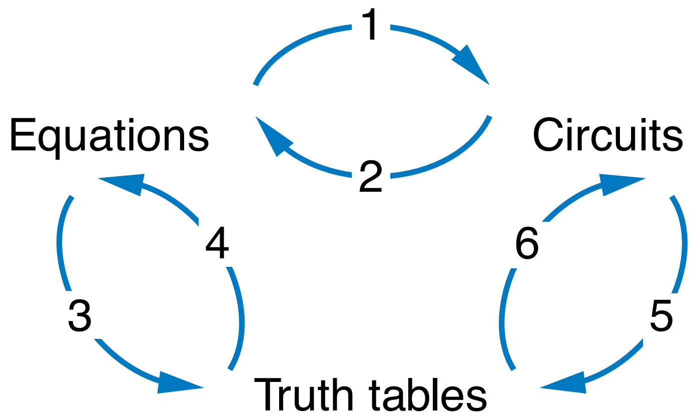
(b)

a	b	c	d	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

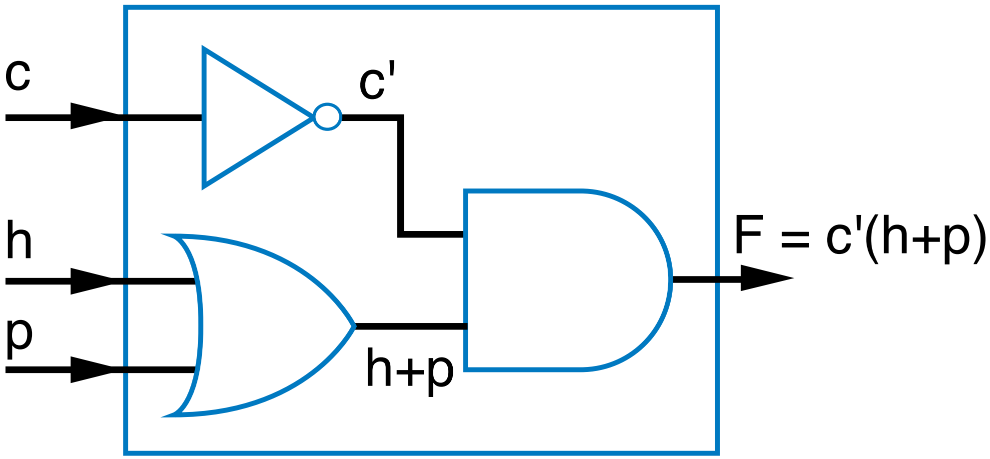
(c)

a	b	c	F
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

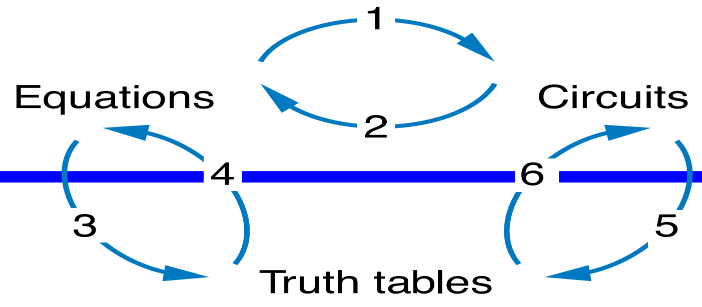
Converting among Representations



- Can convert from any representation to another
- Common conversions
  - Equation to circuit (we did this earlier)
  - Circuit to equation
    - Start at inputs, write expression of each gate output



Converting among Representations



- More common conversions
  - Truth table to equation (which we can then convert to circuit)
    - Easy–just OR each input term that should output 1
  - Equation to truth table
    - Easy—just evaluate equation for each input combination (row)
    - Creating intermediate columns helps

Inputs		Outputs	Term
a	b	F	F = sum of
0	0	1	a'b'
0	1	1	a'b
1	0	0	
1	1	0	

F = a'b' + a'b

Q: Convert to equation

a	b	c	F	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	ab'c
1	1	0	1	abc'
1	1	1	1	abc

F = ab'c + abc' + abc

Q: Convert to truth table: F = a'b' + a'b

Inputs				Output
a	b	a' b'	a' b	F
0	0	1	0	**1**
0	1	0	1	**1**
1	0	0	0	**0**
1	1	0	0	**0**

Example: Converting from Truth Table to Equation

- Parity bit: Extra bit added to data, intended to enable detection of error (a bit changed unintentionally)
  - e.g., errors can occur on wires due to electrical interference
- Even parity: Set parity bit so total number of 1s (data + parity) is even
  - e.g., if data is 001, parity bit is 1 → 0011 has even number of 1s
- Want equation, but easiest to start from truth table for this example

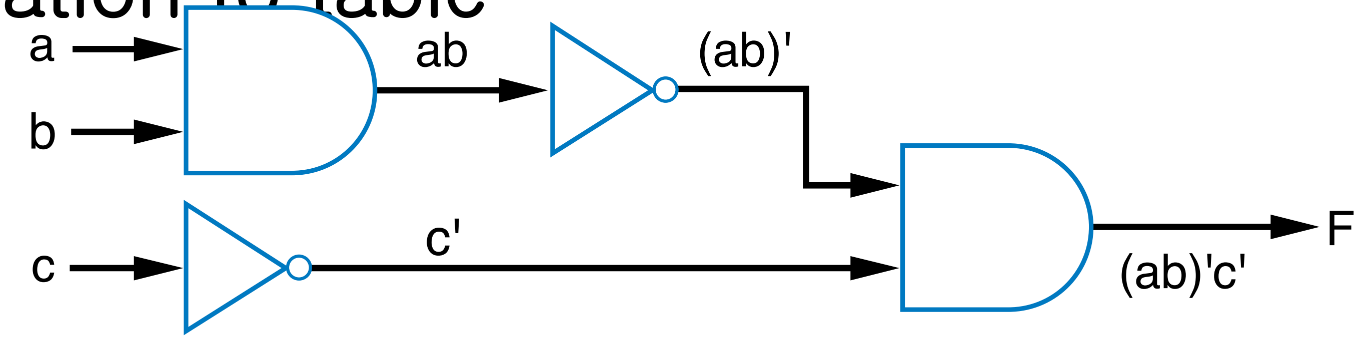
a	b	c	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Convert to eqn.

$$P = a'b'c + a'bc' + ab'c' + abc$$

Example: Converting from Circuit to Truth Table

- First convert to circuit to equation, then equation to table



Inputs						Outputs
a	b	c	ab	(ab)'	c'	F
0	0	0	0	1	1	1
0	0	1	0	1	0	0
0	1	0	0	1	1	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	1	0	1	0
1	1	1	1	0	0	0