SER 232

Computer Systems Fundamentals I

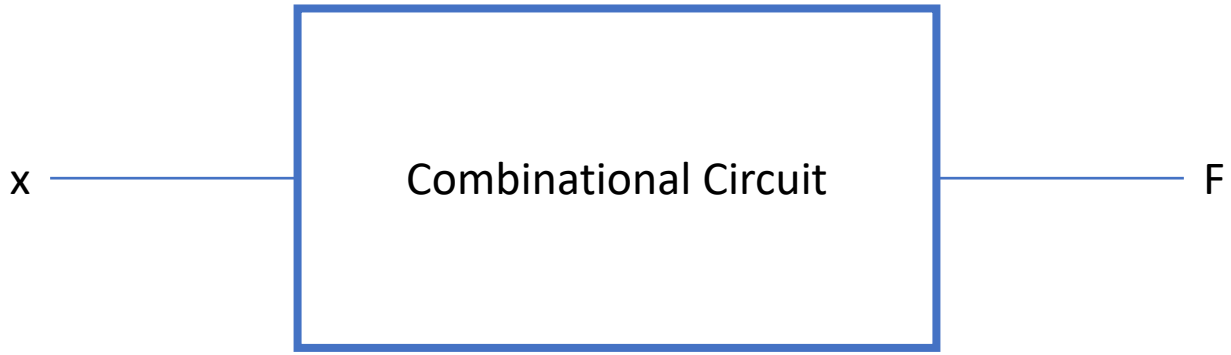
Dr. Heinrichs

Topics

- Combinational Circuit
- Truth Table
- Logic Gates

Combinational Circuit

- A circuit for which the output solely depends on the current input values is called a “combinational circuit”
  - “Outputs are defined by a pure function only based on the current input values”
- Example: Output **F** only depends on current value of input **x**



Truth Table

- Name comes from Boolean values, true and false
- Digital combinational circuits are similar to mathematical functions (or functions in a program)
  - Takes a set of one or more **inputs**
  - For every possible input value there is an **output**
  - All inputs and outputs are digital (1/true or 0/false)

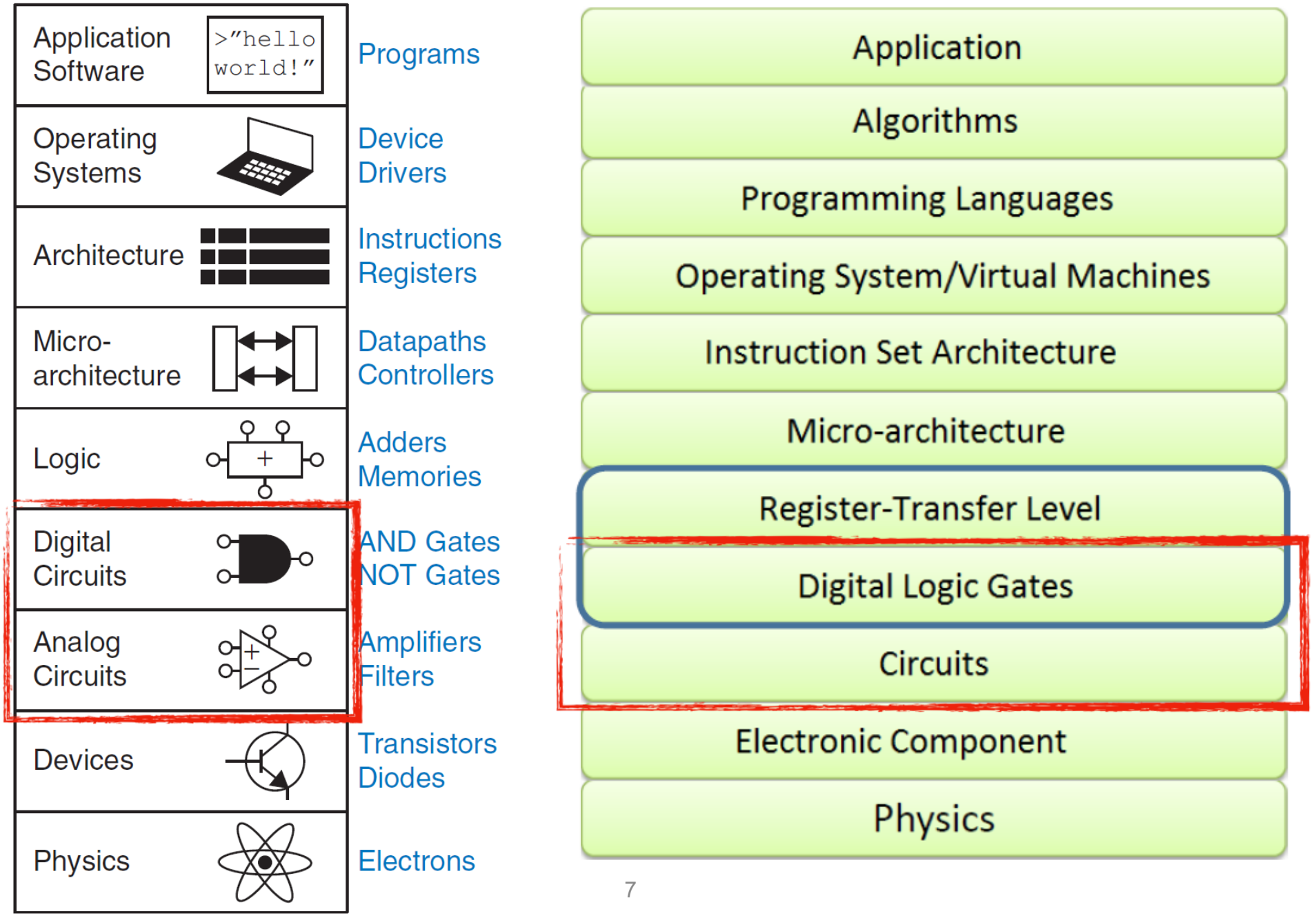
Truth Table

- Table used to describe behavior of circuit
- No knowledge about actual circuit
- Only “what goes in” (input) & “what comes out” (output) visible
  - Also called “blackbox”
- Example: x, y are inputs; F is output

x	y	F

Digital Logic Gates

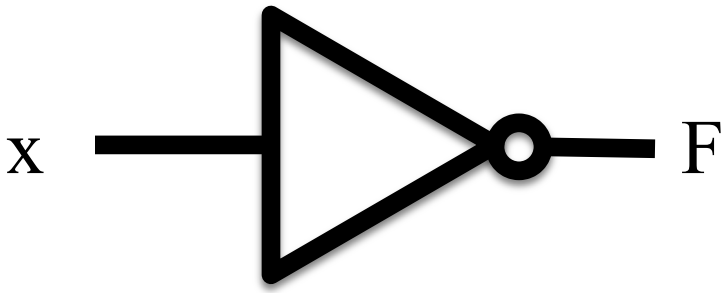
- Abstraction from circuits


Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons
Application
Algorithms
Programming Languages
Operating System/Virtual Machines
Instruction Set Architecture
Micro-architecture
Register-Transfer Level
Digital Logic Gates
Circuits
Electronic Component
Physics

Digital Logic Gates

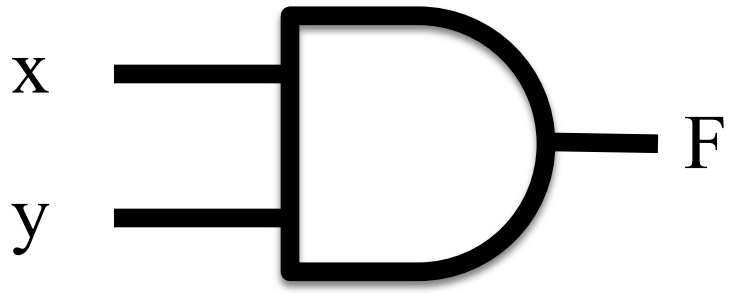
- Abstraction from circuits
  - No need to worry about how analog circuits work
  - Focus on functionality of collection of transistors
    - Logic gates are made out of transistors
  - Wires values are digital (on/1/high or off/0/low)

NOT Gate



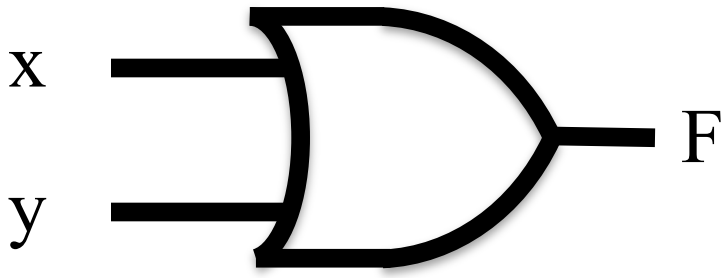
x	F
0	1
1	0

AND Gate



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

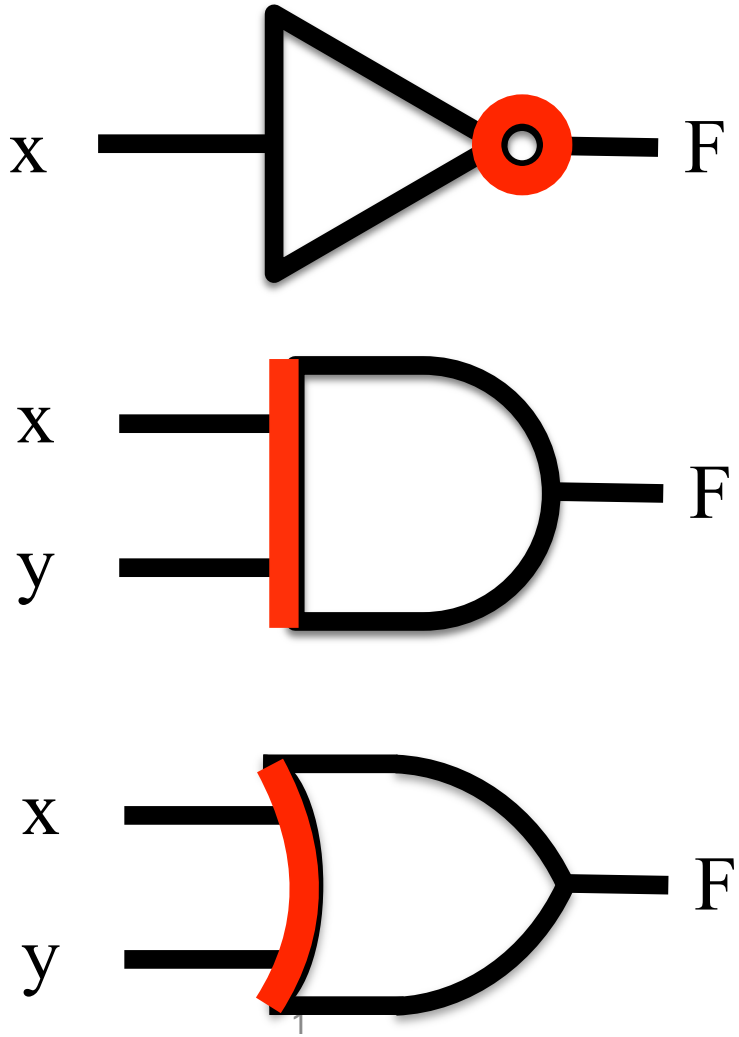
OR Gate



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

Logic Gates

- **NOT**
- **AND**
- **OR**



x	F (NOT)
0	1
1	0

x	y	F (AND)
0	0	0
0	1	0
1	0	0
1	1	1

x	y	F (OR)
0	0	0
0	1	1
1	0	1
1	1	1

SER 232

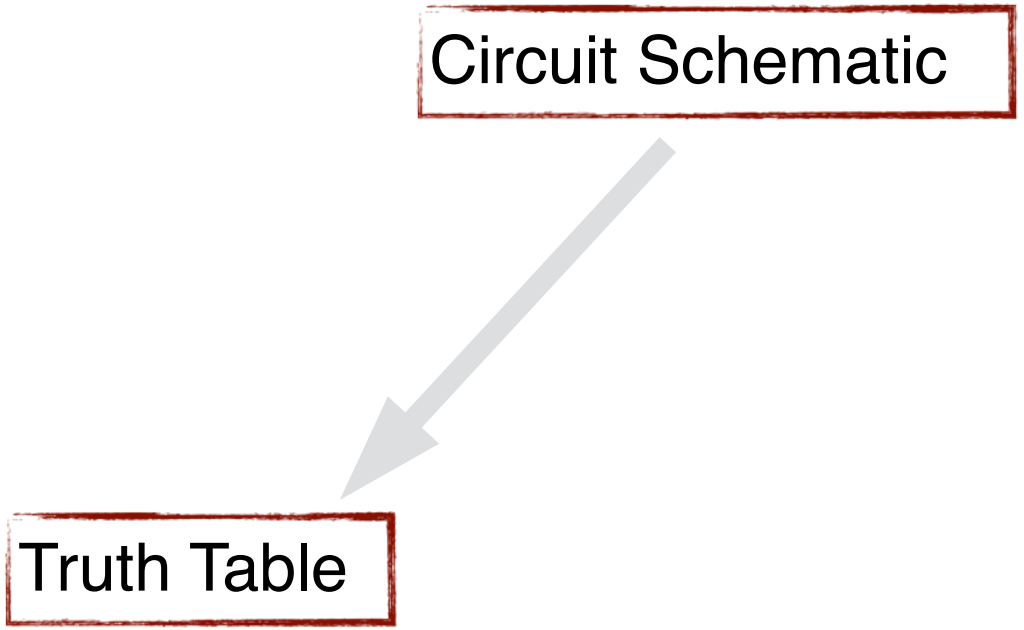
Computer Systems Fundamentals I

Dr. Heinrichs

Topics

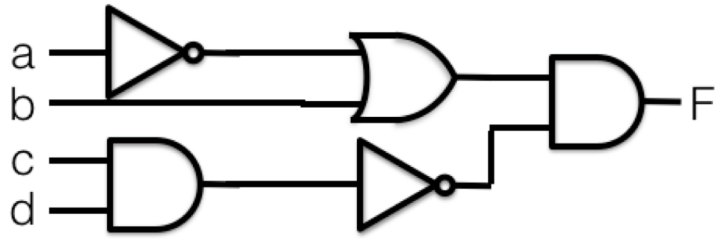
- Circuit Representations
- Circuit Conversion Part 1

Converting Representations



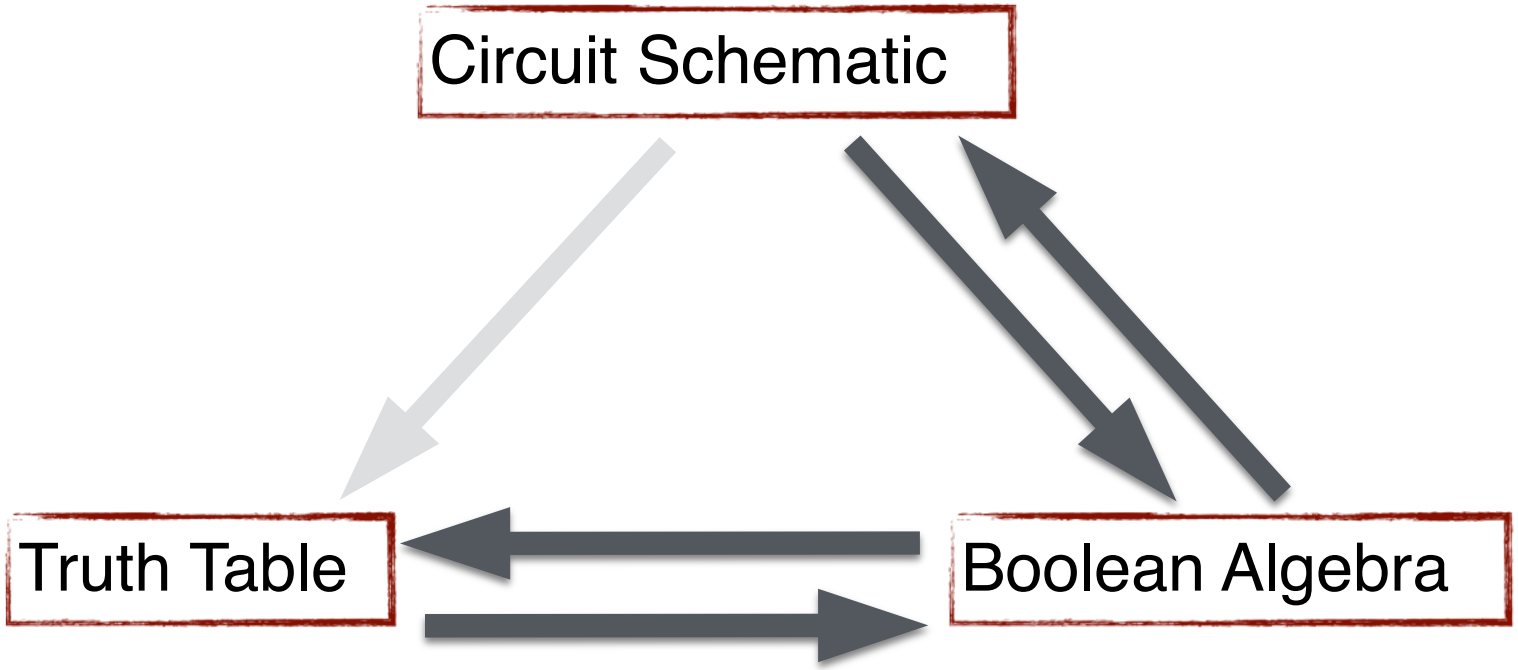
Boolean Algebra

- Ways to represent digital logic gates:
  1. Circuit Schematic
     - Graphical representation
  2. Truth Table
     - Tabular representation
  3. Boolean Algebra
     - Mathematical representation



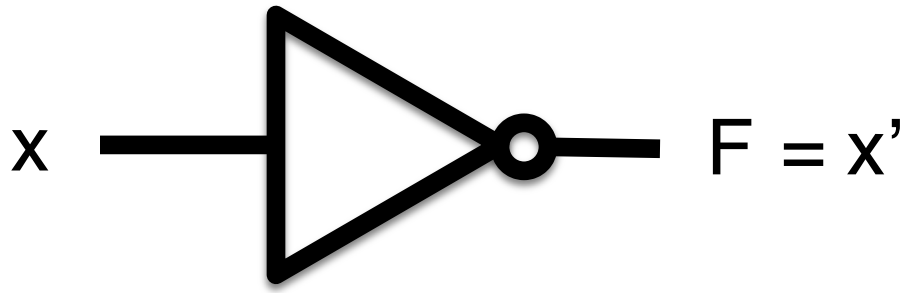
x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

Converting Representations



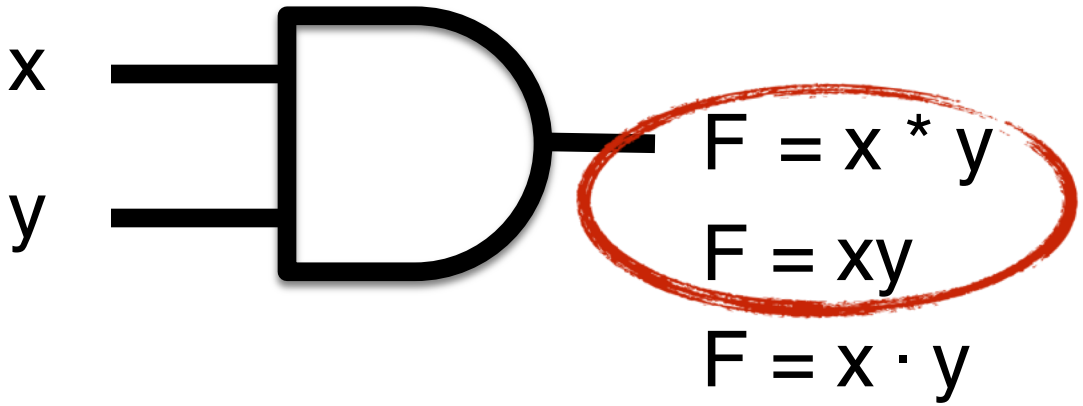
Notation

- NOT Gate



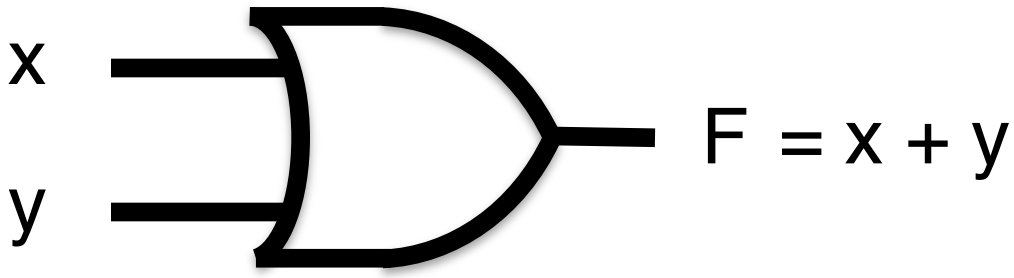
Notation

- AND Gate

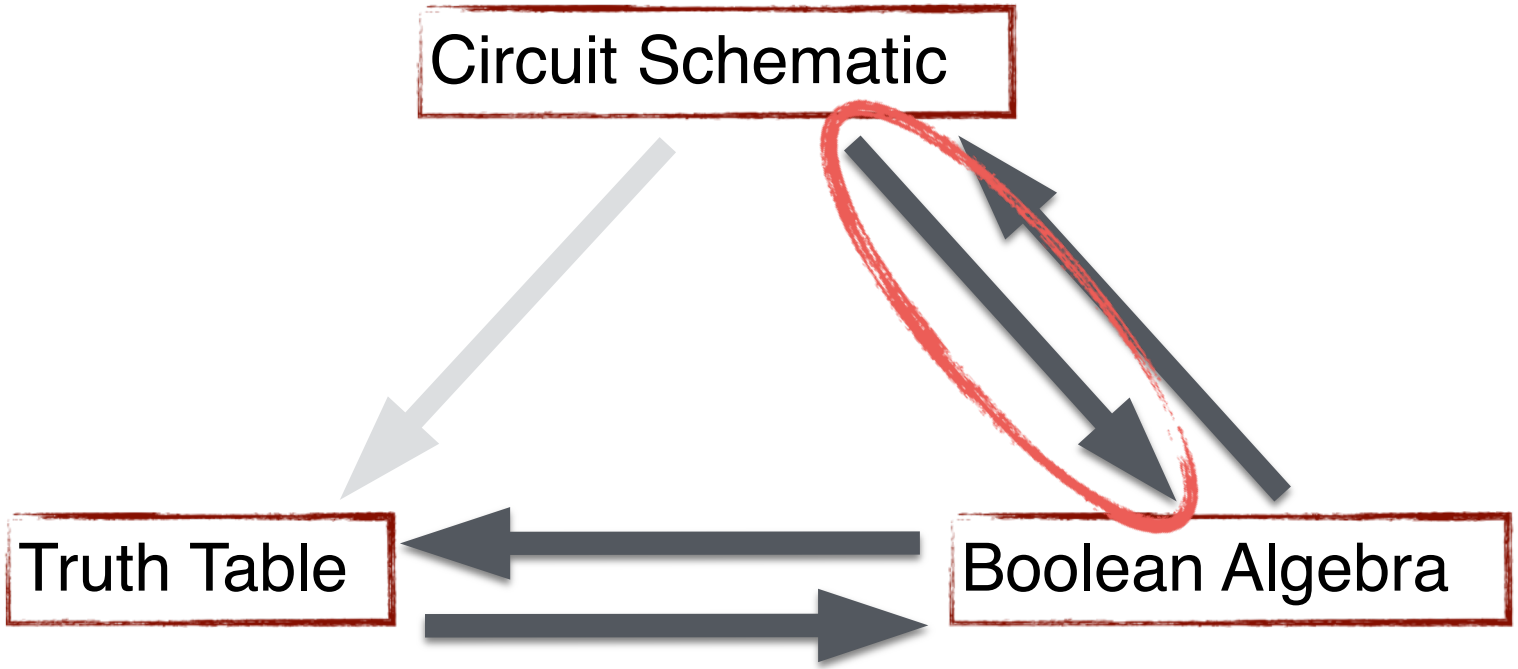


Notation

- OR Gate

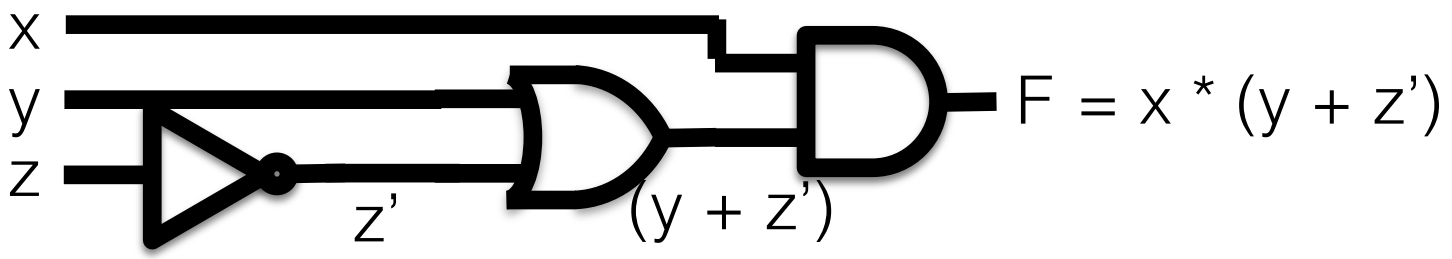


Converting Representations



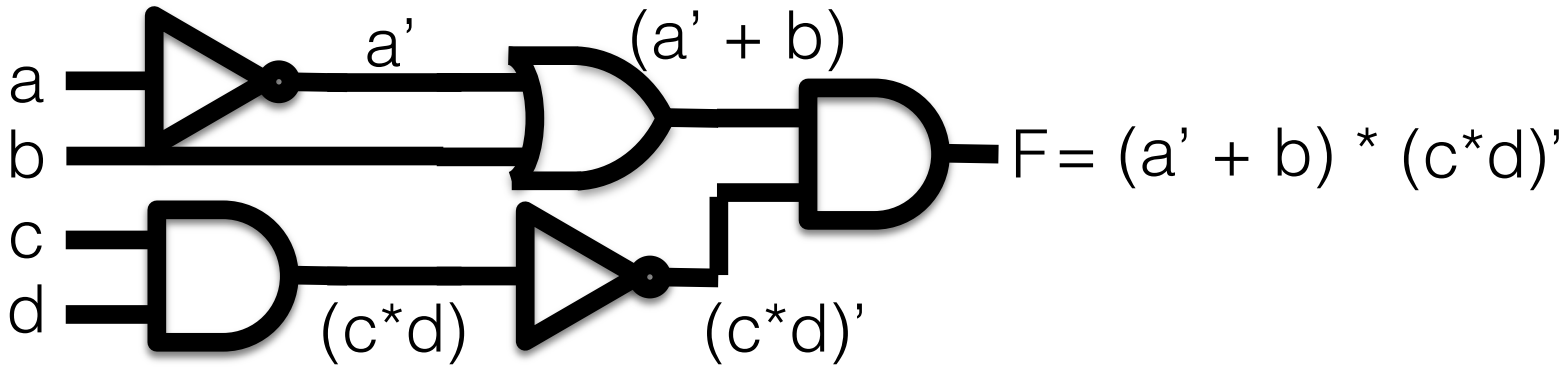
Conversion

- Conversion: Digital Circuit to Boolean Equation

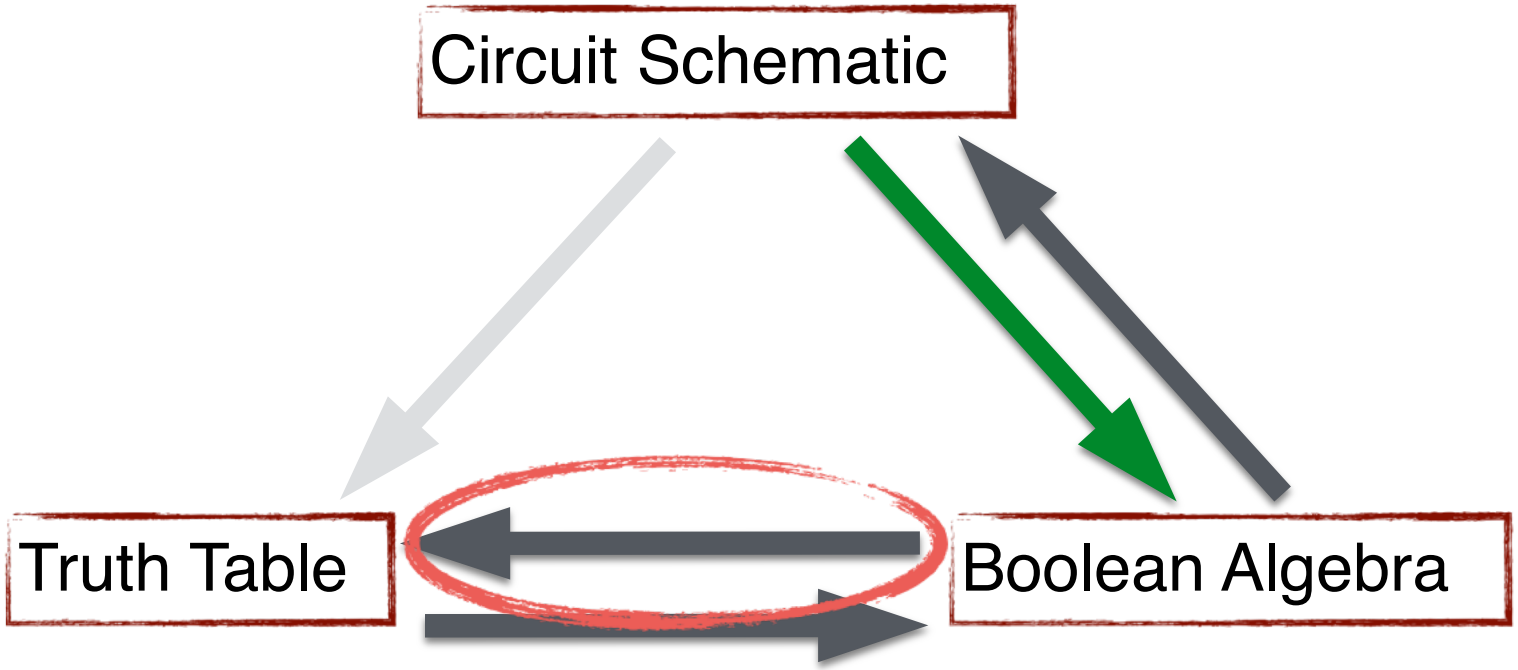


Conversion: Your Turn!

- Conversion: Digital Circuit to Boolean Equation

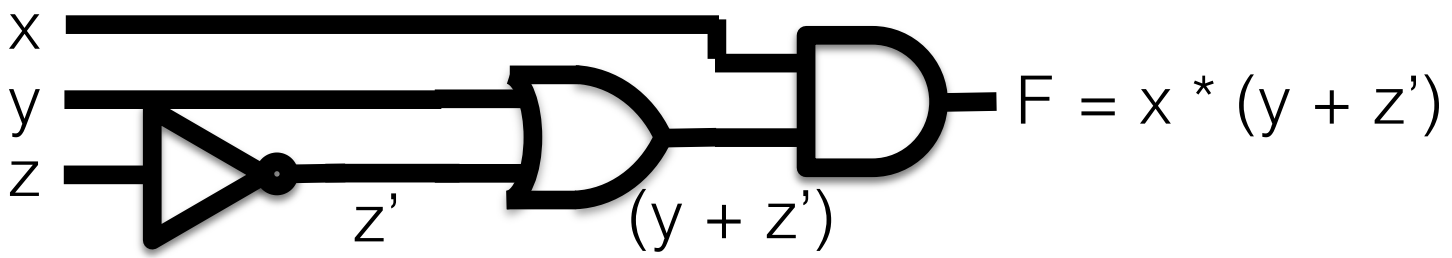


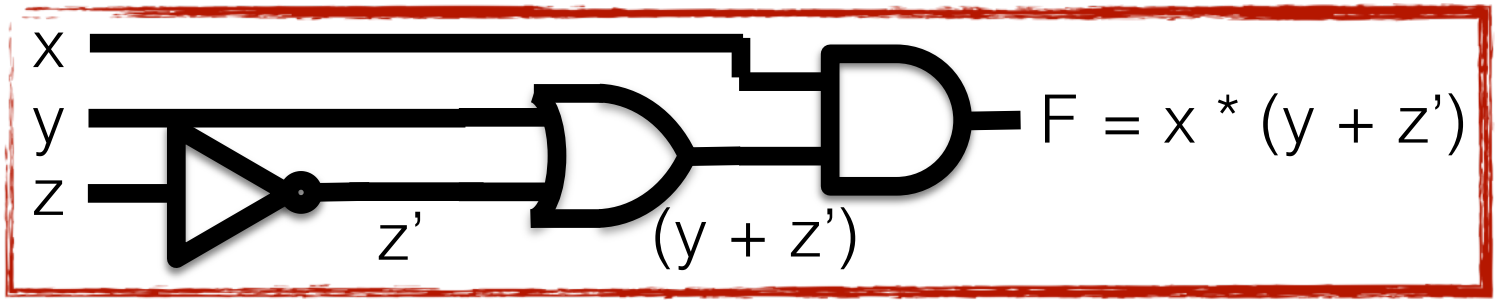
Converting Representations



Conversion

- Conversion: Boolean Equation to Truth Table





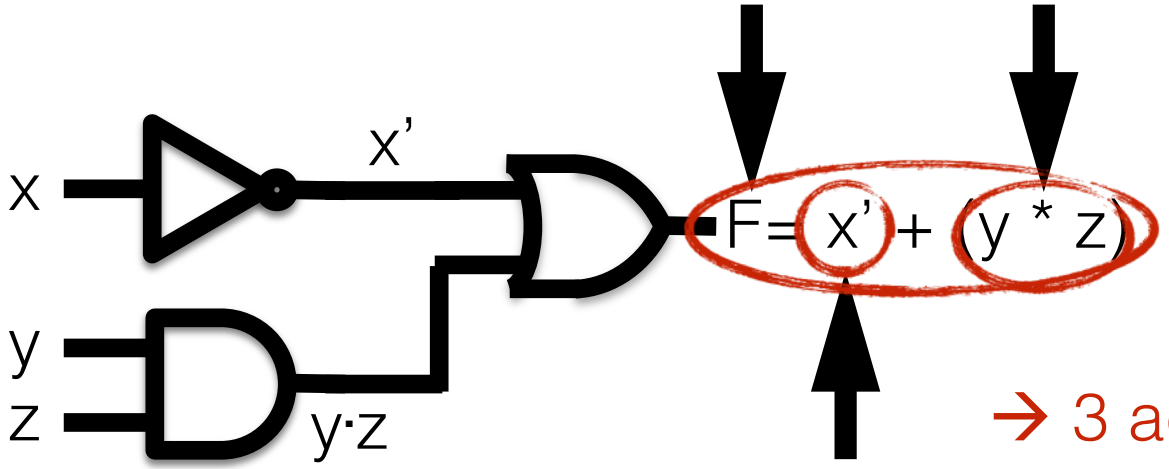
- Conversion: Boolean Equation to Truth Table

x	y	z	z'	y + z'	F = x * (y + z')
0	0	0	1	1	0
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	1	1

$F = x * (y + z')$

Conversion: Your Turn!

- Conversion: Boolean Equation to Truth Table



→ 3 additional columns!

Conversion: Your Turn!

- Conversion: Boolean Equation to Truth Table

x	y	z	x’	y * z	F = x’ + (y * z)
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

F= x’ + (y * z)

SER 232

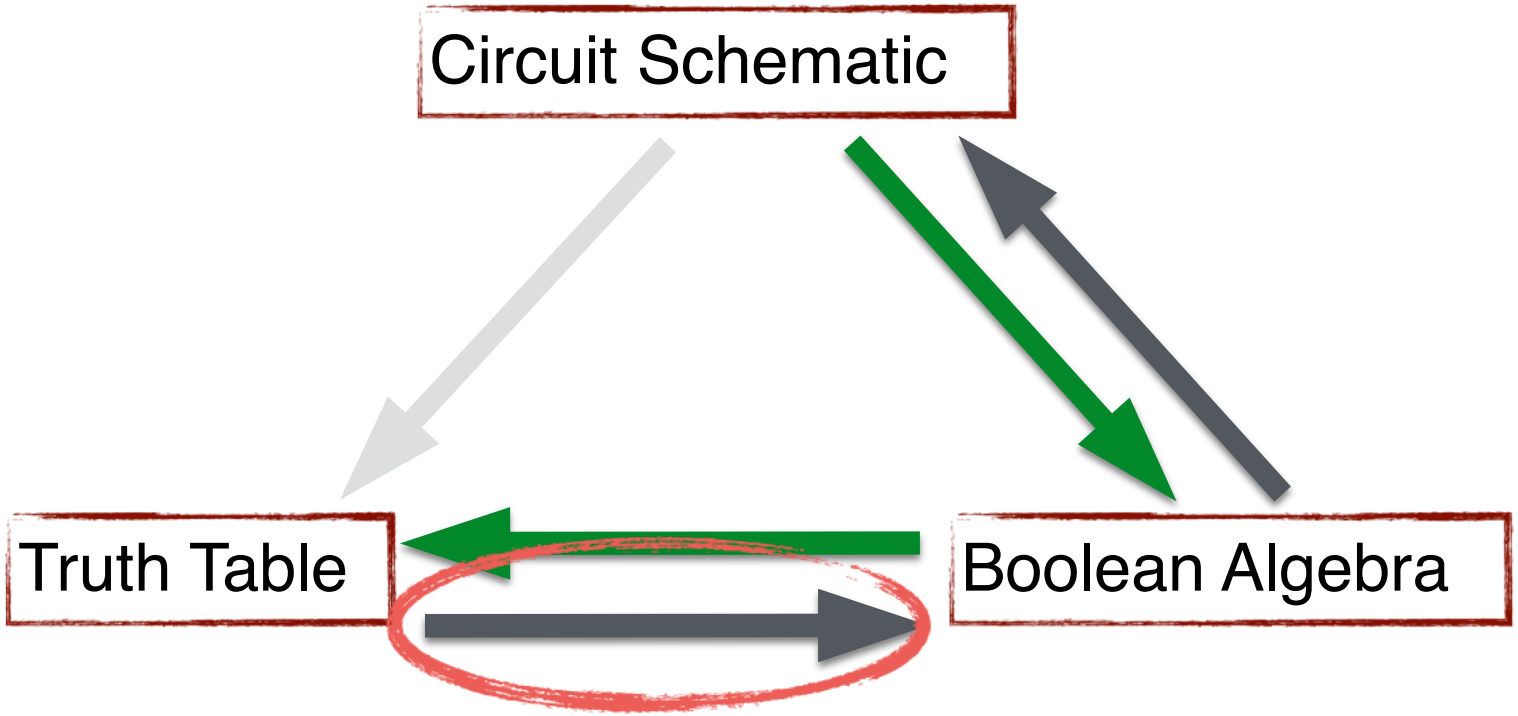
Computer Systems Fundamentals I

Dr. Heinrichs

Topics

- Circuit Conversion Part 2

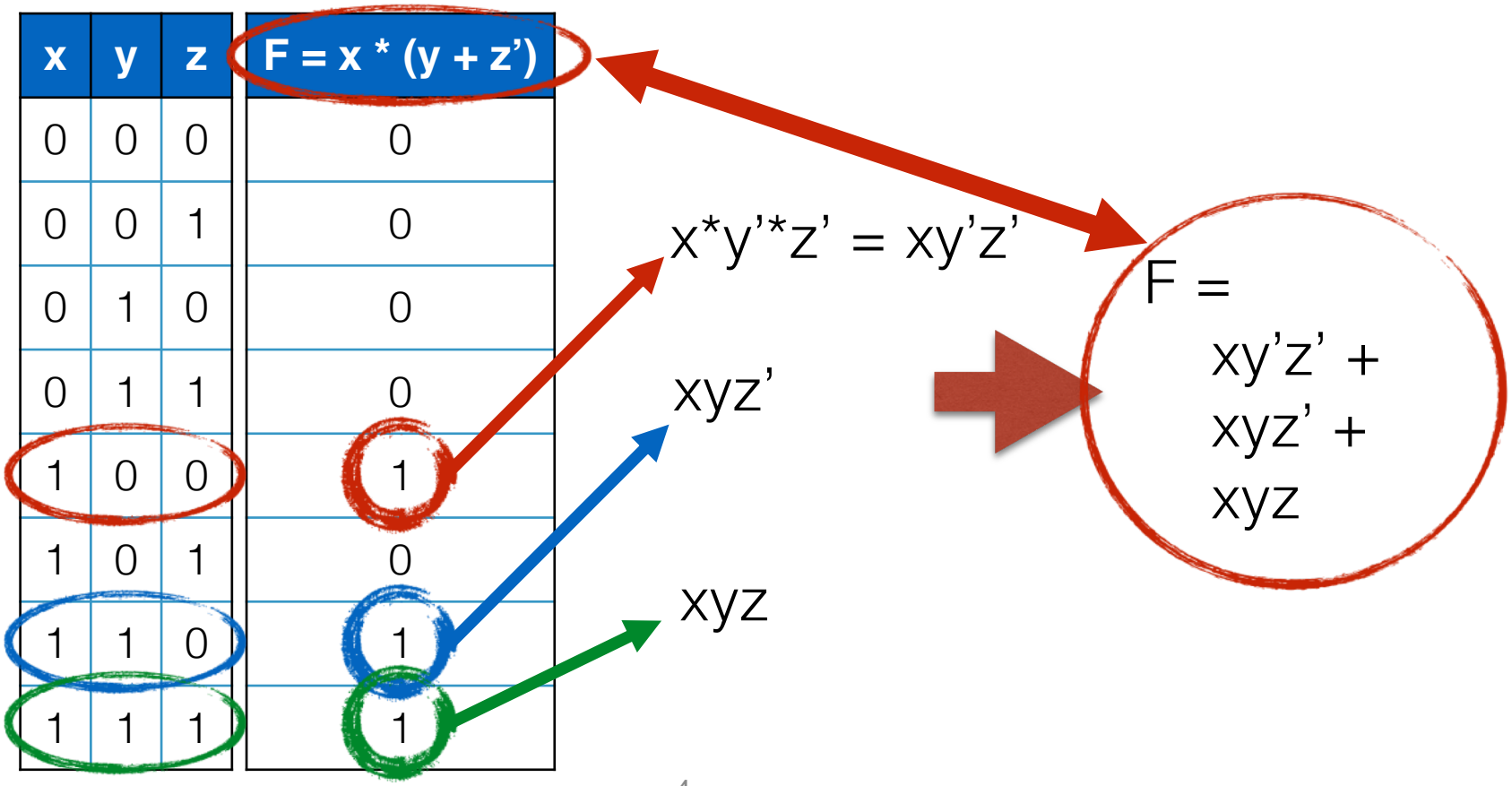
Converting Representations



Conversion

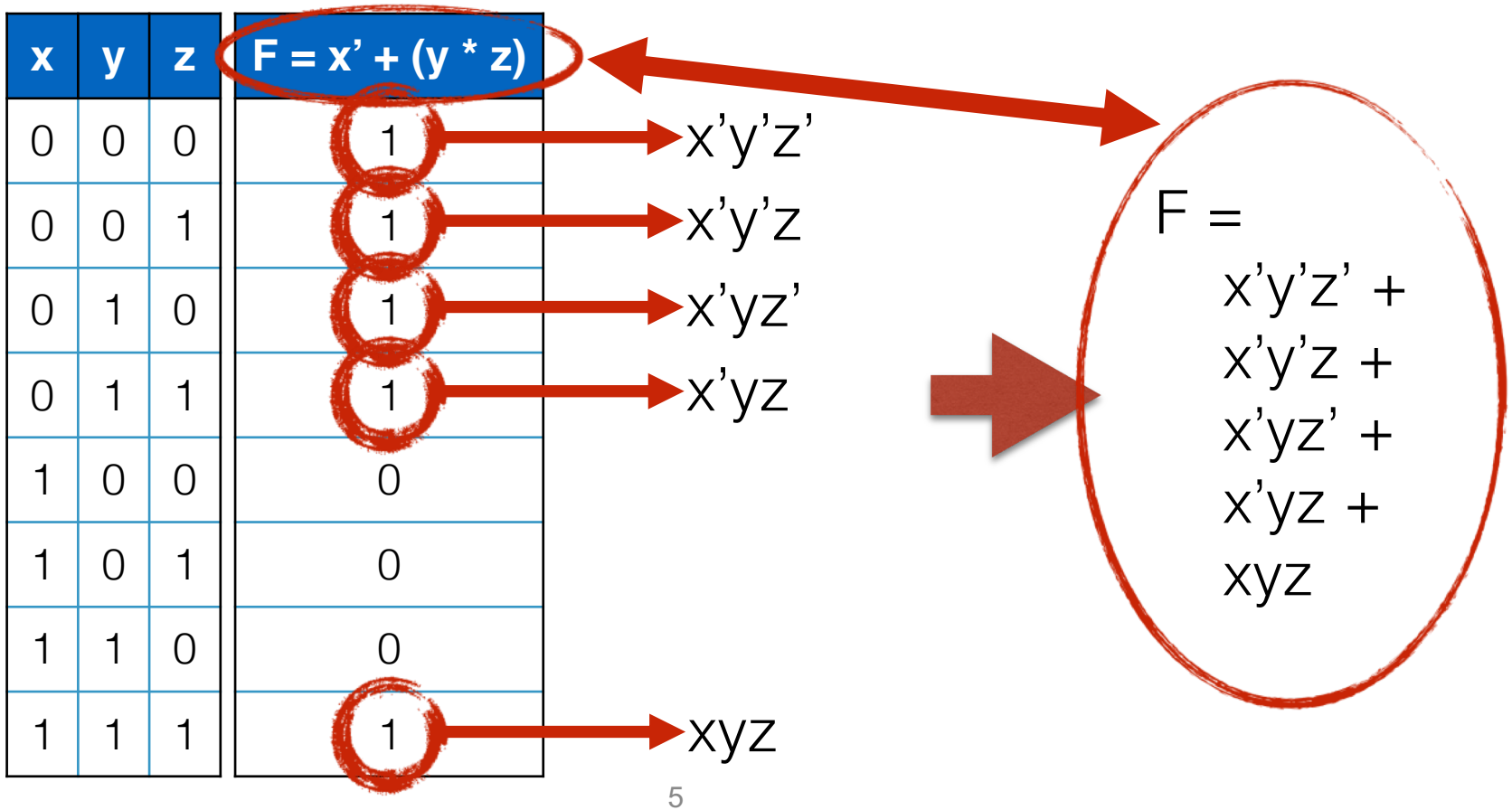
- Conversion: Truth Table to Boolean Equation

x	y	z	F = x * (y + z')
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

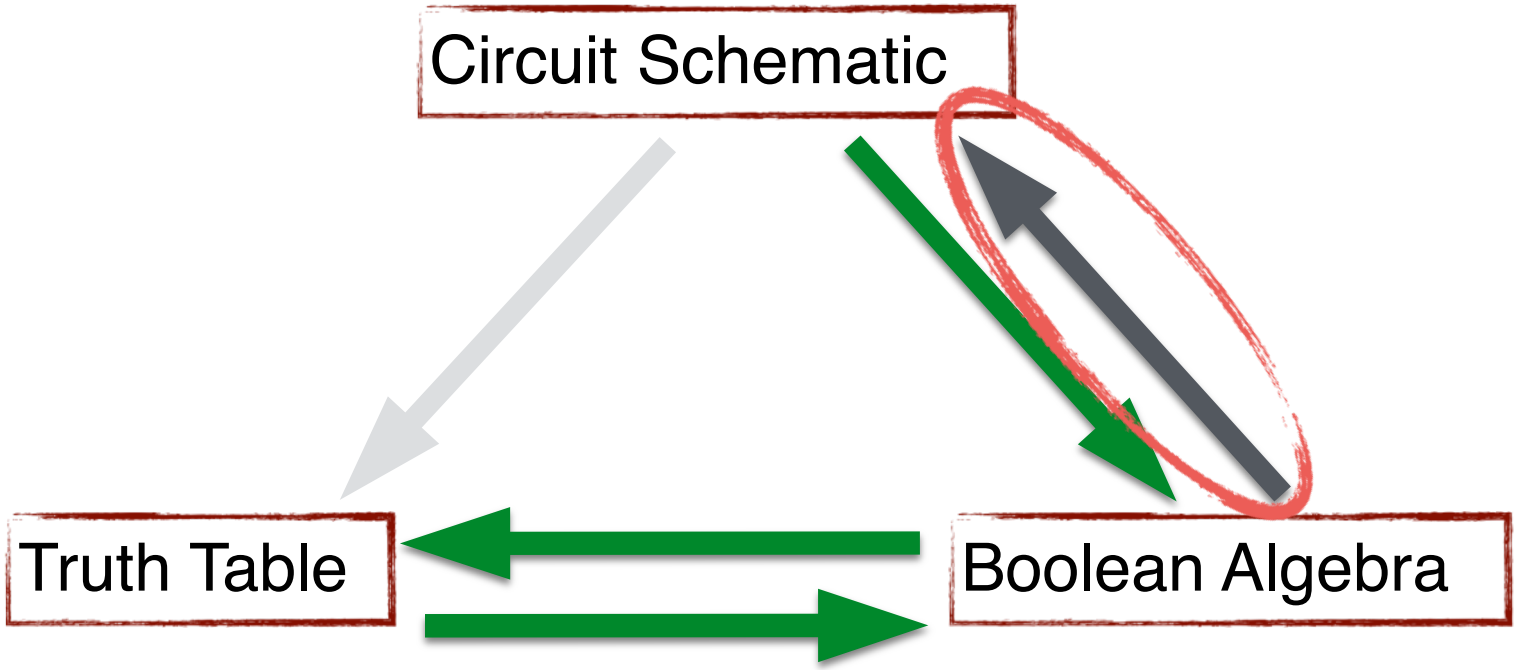


Conversion: Your Turn!

- Conversion: Truth Table to Boolean Equation



Converting Representations



Conversion

- Conversion: Boolean Algebra to Circuit Schematic

$$F = xy'z' + xyz' + xyz$$

Logic Gates

- Logic Gates with more than 2 inputs
  - AND: only 1 if ALL inputs 1
  - OR: only 0 if ALL input 0

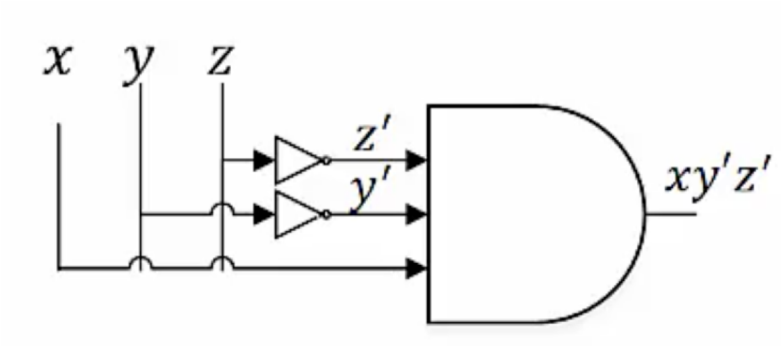
x	y	z	F (AND)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

x	y	z	F (OR)
0	0	0	0
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	1

Conversion

- Conversion: Boolean Algebra to Circuit Schematic

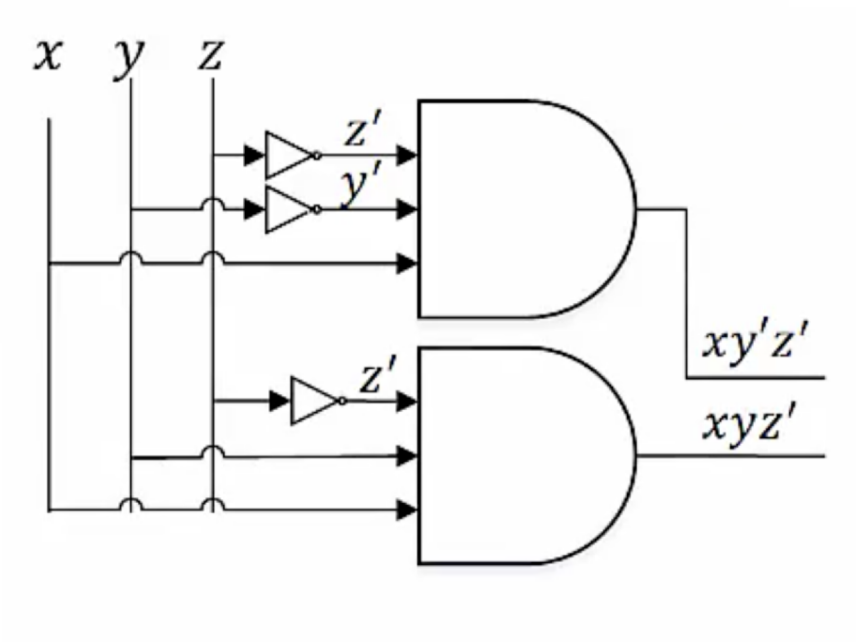
F = **xy'z'** + xyz' + xyz



Conversion

- Conversion: Boolean Algebra to Circuit Schematic

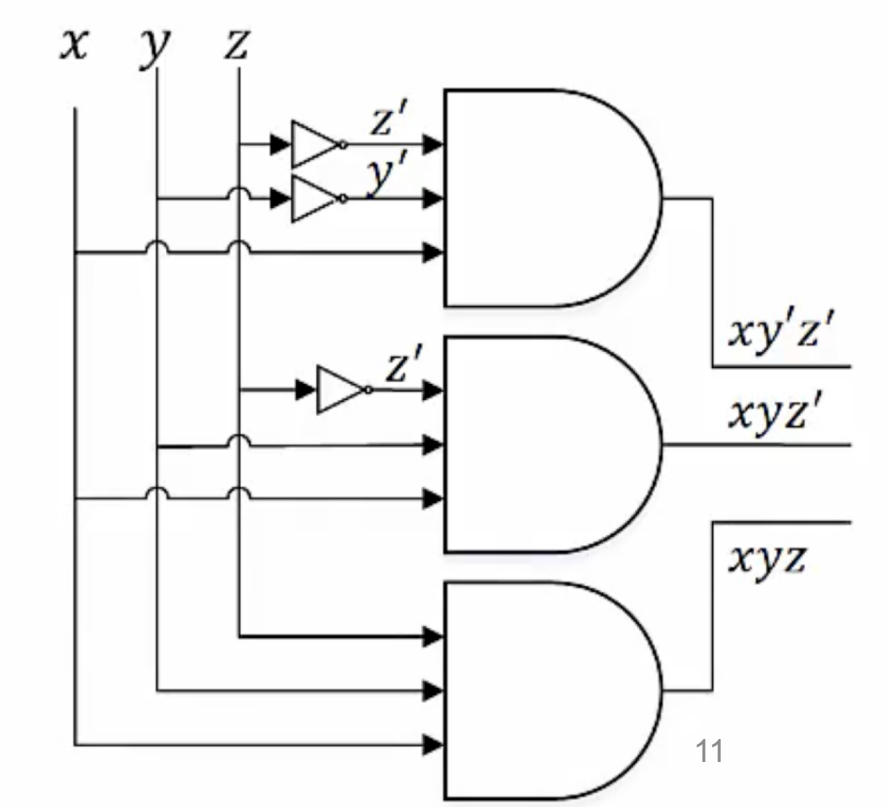
F = xy'z' + **xyz'** + xyz



Conversion

- Conversion: Boolean Algebra to Circuit Schematic

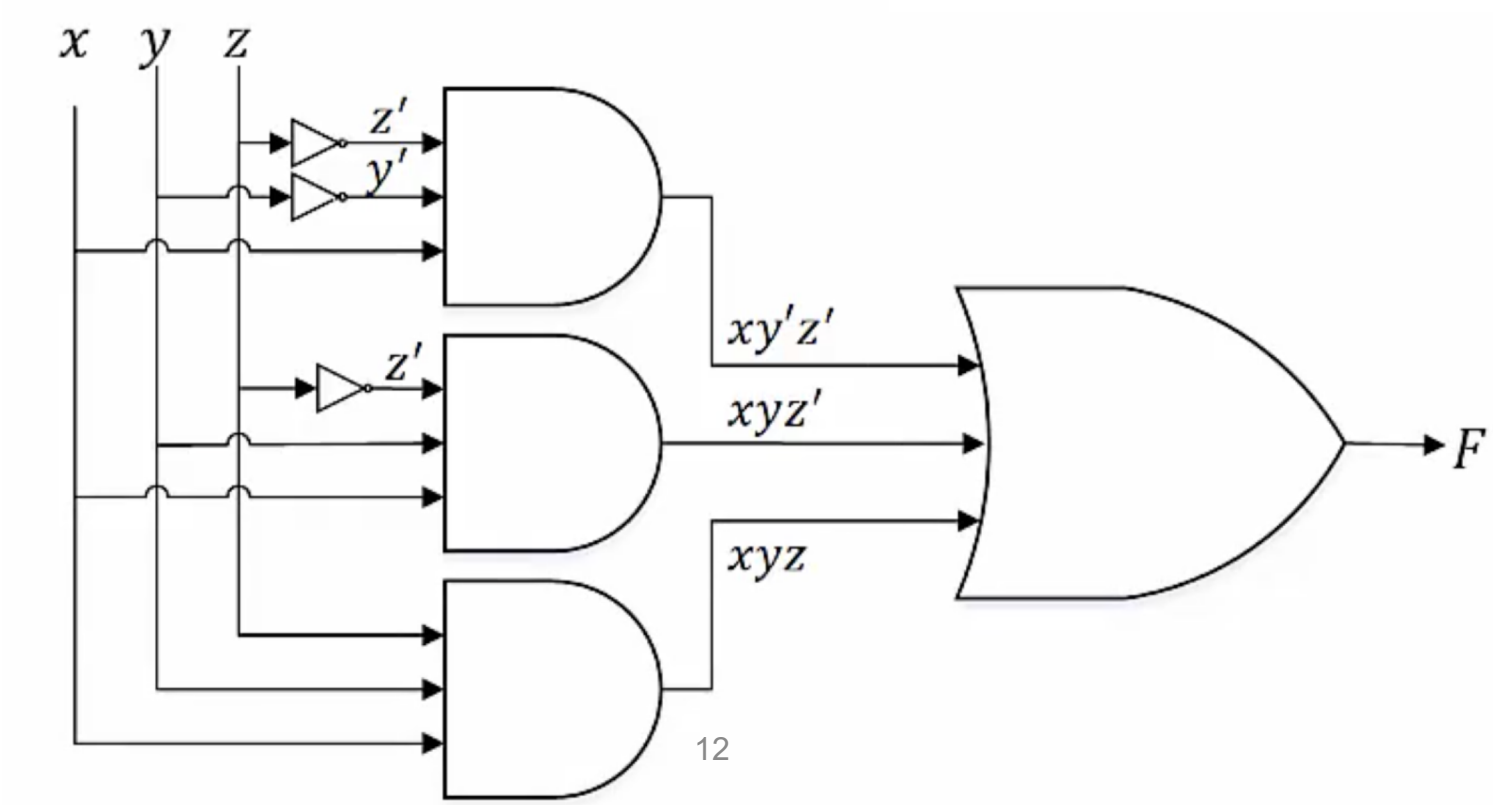
$$F = xy'z' + xyz' + \mathbf{xyz}$$



Conversion

- Conversion: Boolean Algebra to Circuit Schematic

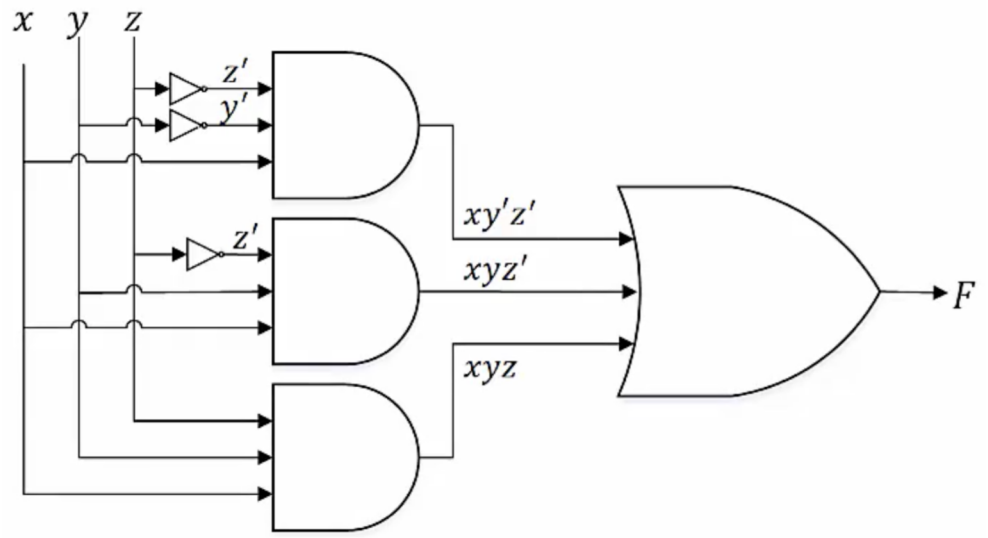
$$F = xy'z' + xyz' + xyz$$

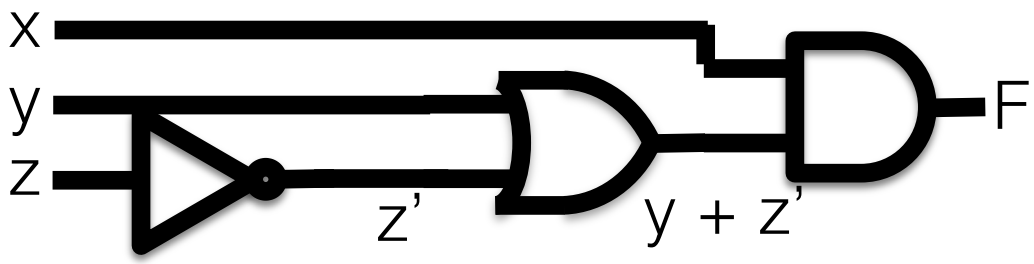


Conversion

- Conversion: Boolean Algebra to Circuit Schematic
  - Equivalence can be shown with Boolean Algebra
  - Alternative: Create and compare Truth Tables

F = xy'z' + xyz' + xyz = F = x (y + z')

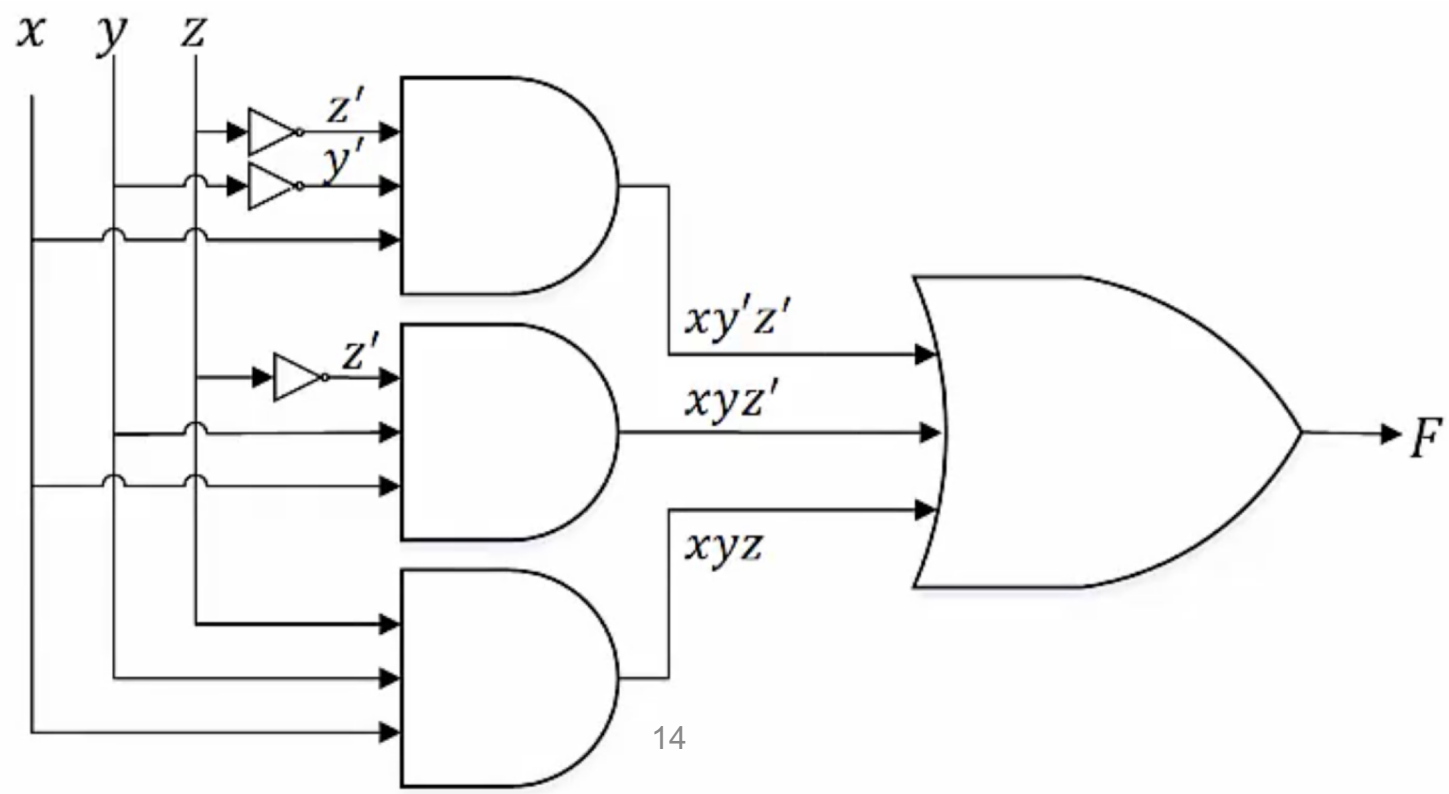




Conversion: Your Turn!

- Create Truth Table for this circuit schematic

$$F = xy'z' + xyz' + xyz$$



Conversion: Your Turn!

- Create Truth Table for this circuit schematic

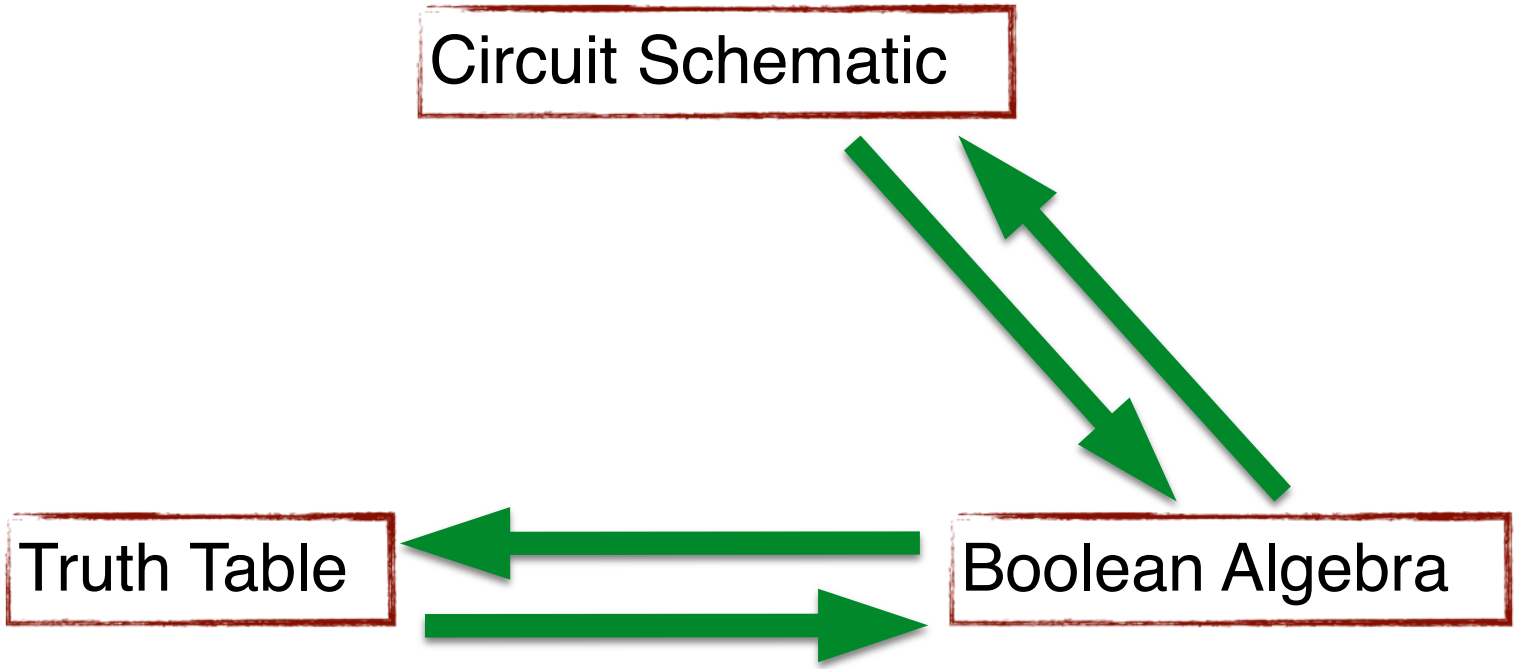
$$F = xy'z' + xyz' + xyz$$

x	y	z	y'	z'	xy'z'	xyz'	xyz	F
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	1	1	0	0	1
1	0	1	1	0	0	0	0	0
1	1	0	0	1	0	1	0	1
1	1	1	0	0	0	0	1	1

=

F = x * (y + z')
0
0
0
0
1
0
1
1

Converting Representations



SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

Topics

- Boolean Algebra Part 1

Boolean Algebra

- Can be used to:
  - Verify that two different equations describe same behavior
  - Simplify Boolean equations
    - Make it easier to create circuits and truth tables from it

Prove Equivalence

- Converting circuit to truth table & back two different Boolean equations resulted:
  - F = x * (y + z')
  - F = x*y'*z' + x*y*z' + x*y*z
- Proving equivalence with Boolean algebra:

F = x*y'*z' + x*y*z' + x*y*z

F = x * (y'*z' + y*z' + y*z) *(distributive property)*

Distributivity

- Distributive property (of AND over OR)
  → `a * (b + c) = a * b + a * c`
- Distributive property (of OR over AND)
  → `a + (b * c) = (a + b) * (a + c)`

Prove Equivalence

- Prove equation below is equal to: F = x * (y + z')

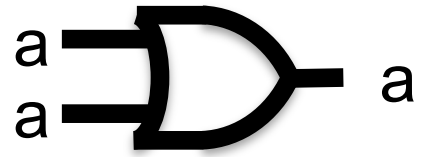
F = x*y'*z' + x*y*z' + x*y*z

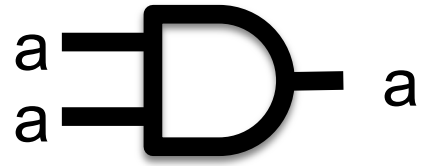
F = x * (y'*z' + y*z' + y*z) *(distributive property)*

F = x * (y'*z' + y*z' + y*z' + y*z) *(idempotence prop.)*

Idempotence

- Idempotence property
  - → `a + a = a`
  - → `a * a = a`





Prove Equivalence

- Prove equation below is equal to: F = x * (y + z')

F = x*y'*z' + x*y*z' + x*y*z

F = x * (y'*z' + y*z' + y*z) *(distributive property)*

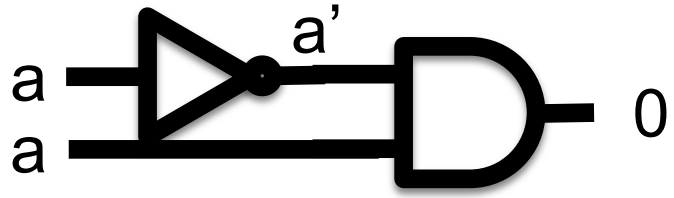
F = x * (y'*z' + y*z' + y*z' + y*z) *(idempotence prop.)*

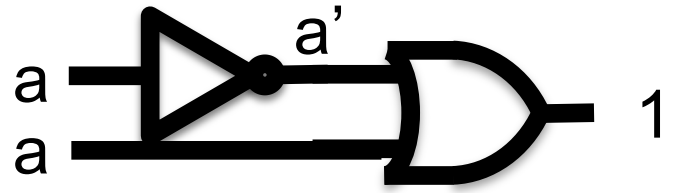
F = x * (z'*(y' + y) + y*(z' + z)) *(distributive property)*

F = x * (z'*(1) + y*(1)) *(complement prop.)*

Complementation

- Complement property 1
  → `a * a' = 0`
- Complement property 2
  → `a + a' = 1`





Prove Equivalence

- Prove equation below is equal to: F = x * (y + z')

F = x*y'*z' + x*y*z' + x*y*z

F = x * (y'*z' + y*z' + y*z) *(distributive property)*

F = x * (y'*z' + y*z' + y*z' + y*z) *(idempotence prop.)*

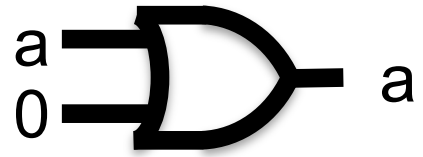
F = x * (z'*(y' + y) + y*(z' + z)) *(distributive property)*

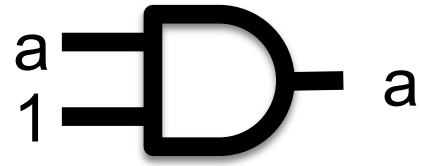
F = x * (z'*(1) + y*(1)) *(complement prop.)*

F = x * (z' + y) *(identity property)*

Identity

- Identity property
  - → `a + 0 = a`
  - → `a * 1 = a`





Prove Equivalence

- Prove equation below is equal to: F = x * (y + z')

F = x*y'*z' + x*y*z' + x*y*z

F = x * (y'*z' + y*z' + y*z) *(distributive property)*

F = x * (y'*z' + y*z' + y*z' + y*z) *(idempotence prop.)*

F = x * (z'*(y' + y) + y*(z' + z)) *(distributive property)*

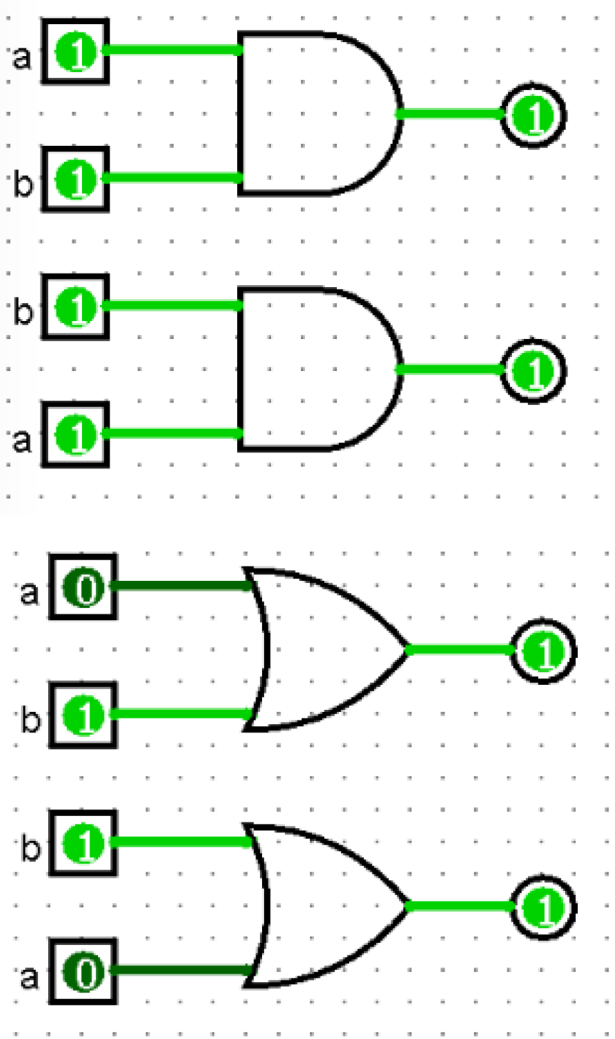
F = x * (z'*(1) + y*(1)) *(complement prop.)*

F = x * (z' + y) *(identity property)*

F = x * (y + z') *(commutativity prop.)*

Commutativity

- Commutative property
  - → `a + b = b + a`
  - → `a * b = b * a`



Prove Equivalence

- Prove equation below is equal to: F = x * (y + z')

Equivalent!

F = x*y'*z' + x*y*z' + x*y*z

F = x * (y'*z' + y*z' + y*z) *(distributive property)*

F = x * (y'*z' + y*z' + y*z' + y*z) *(idempotence prop.)*

F = x * (z'*(y' + y) + y*(z' + z)) *(distributive property)*

F = x * (z'*(1) + y*(1)) *(complement prop.)*

F = x * (z' + y) *(identity property)*

F = x * (y + z') *(commutativity prop.)*

SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

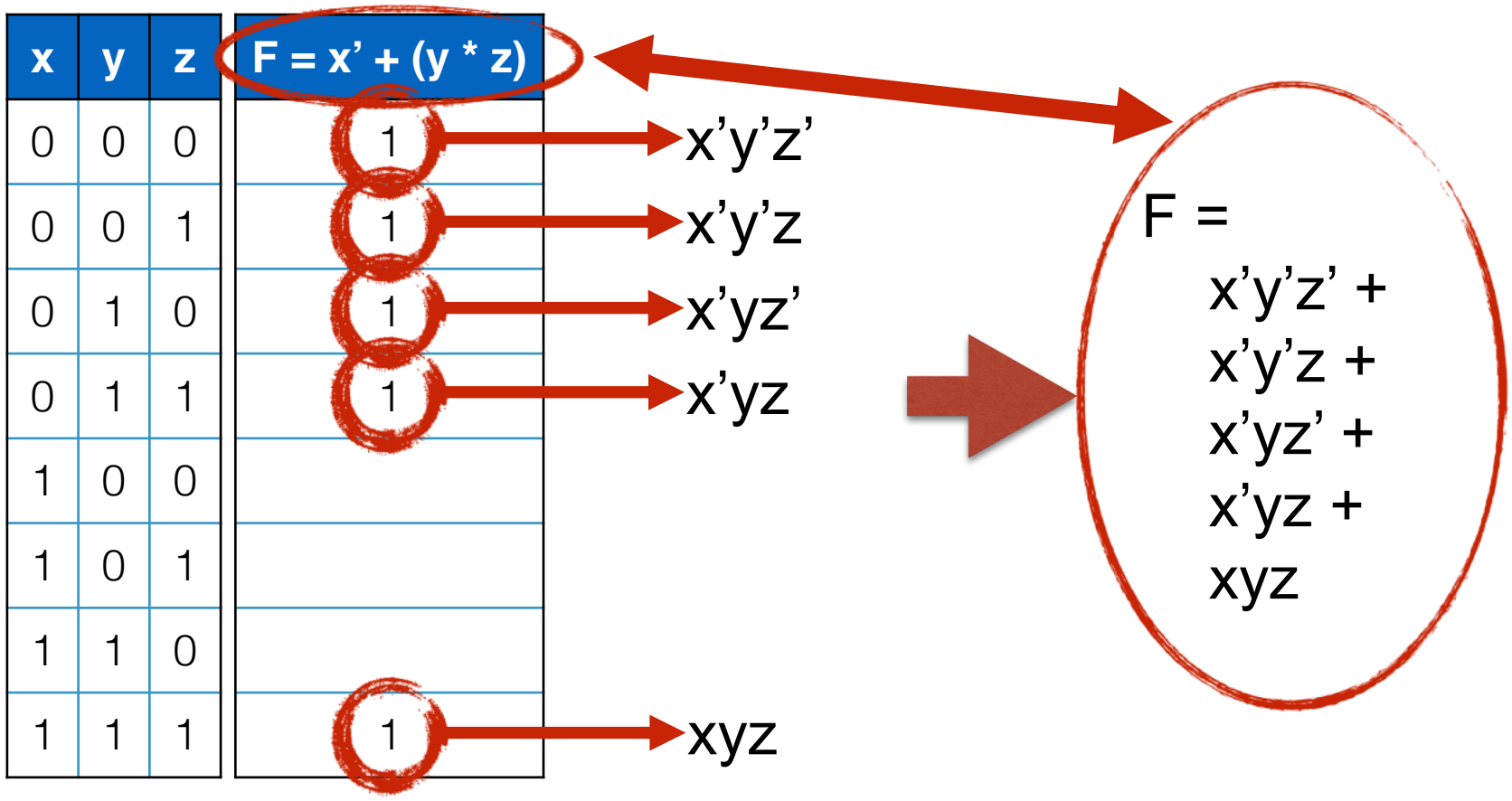
Topics

- Boolean Algebra Part 2

Boolean Algebra

- Covered laws so far:
  - Distributivity
  - Idempotence
  - Complementation
  - Identity
  - Commutativity

Boolean Algebra



Boolean Algebra

- Making sure equation is equal to: F = x' + (y * z)

F = x'*y'*z' + x'*y'*z + x'*y*z' + x'*y*z + x*y*z

F = x'*(y'*z' + y'*z + y*z' + y*z) + x*y*z *(distributive property)*

F = x'*(y'*(z' + z) + y*(z' + z)) + x*y*z *(distributive property)*

F = x'*(y'*(1) + y*(1)) + x*y*z *(complement property)*

F = x'*(y' + y) + x*y*z *(identity property)*

F = x'*(1) + x*y*z *(complement property)*

F = x' + x*y*z *(identity property)*

F = (x' + x) * (x' + y) * (x' + z) *(distributivity property)*

F = (1) * (x' + y) * (x' + z) *(complement property)*

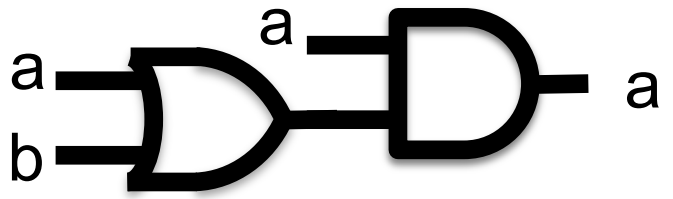
F = (x' + y) * (x' + z) *(identity property)*

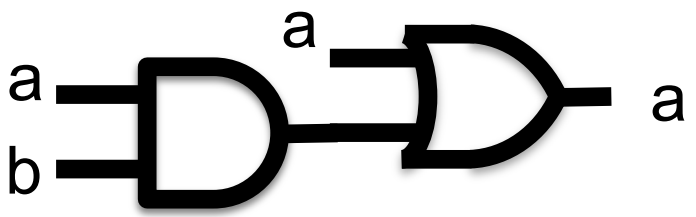
F = (x' + y) * x' + (x' + y) * z *(distributivity property)*

F = x' + (x' + y) * z *(absorption property)*

Absorption

- Absorption property
  - → `a * (a + b) = a`
  - → `a + (a * b) = a`





Boolean Algebra

- Making sure equation is equal to: F = x' + (y * z)

F = x'*y'*z' + x'*y'*z + x'*y*z' + x'*y*z + x*y*z

F = x'*(y'*z' + y'*z + y*z' + y*z) + x*y*z *(distributive property)*

F = x'*(y'*(z' + z) + y*(z' + z)) + x*y*z *(distributive property)*

F = x'*(y'*(1) + y*(1)) + x*y*z *(complement property)*

F = x'*(y' + y) + x*y*z *(identity property)*

F = x'*(1) + x*y*z *(complement property)*

F = x' + x*y*z *(identity property)*

F = (x' + x) * (x' + y) * (x' + z) (distributivity property)

F = (1) * (x' + y) * (x' + z) *(complement property)*

F = (x' + y) * (x' + z) *(identity property)*

F = (x' + y) * x' + (x' + y) * z *(distributivity property)*

F = x' + (x' + y) * z *(absorption property)*

F = x' + x'*z + y*z *(distributivity property)*

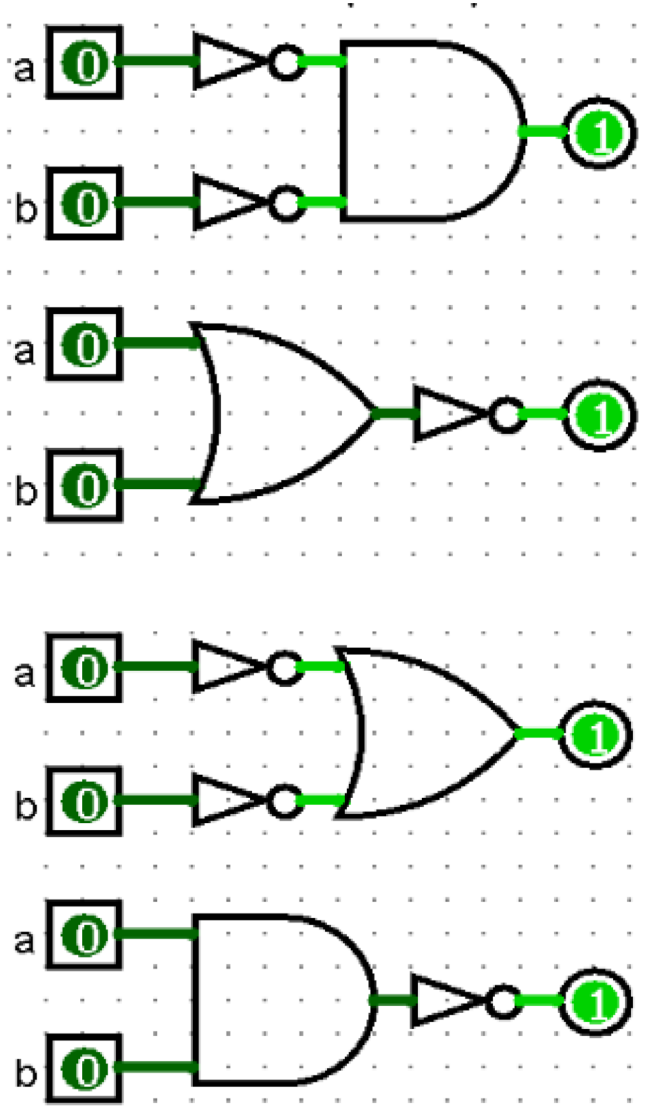
F = x' + (y * z) *(absorption property)*

Boolean Algebra Laws

- Covered
  - Distributivity
  - Idempotence
  - Complementation
  - Identity
  - Commutativity
  - Absorption
- Remaining:
  - De Morgan
  - Associativity
  - Double Negation
  - Annihilator

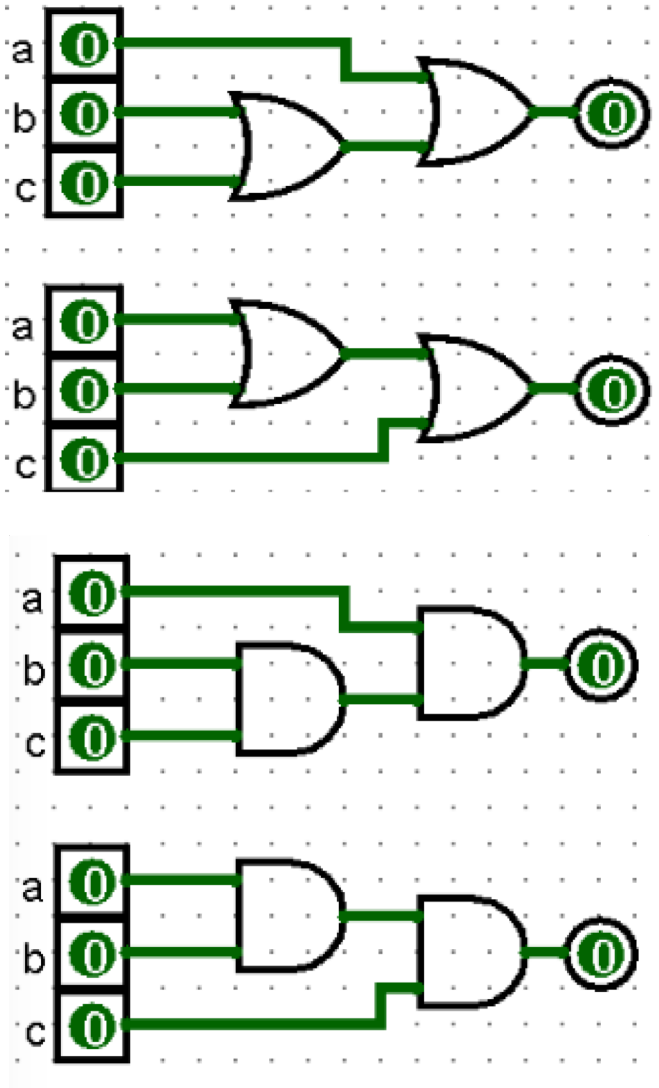
De Morgan

- De Morgan 1:
  → `a' * b' = (a + b)'`
- De Morgan 2:
  → `a' + b' = (a * b)'`



Associativity

- Associativity of OR:
  → `a+(b+c) = (a+b)+c`
- Associativity of AND:
  → `a*(b*c) = (a*b)*c`

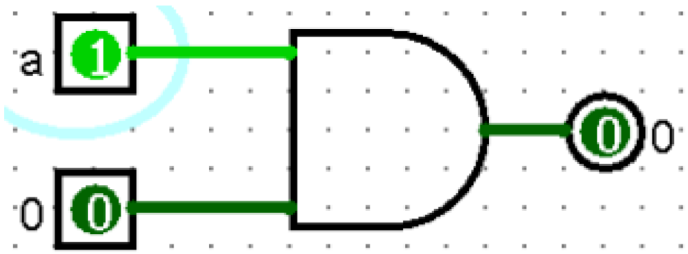


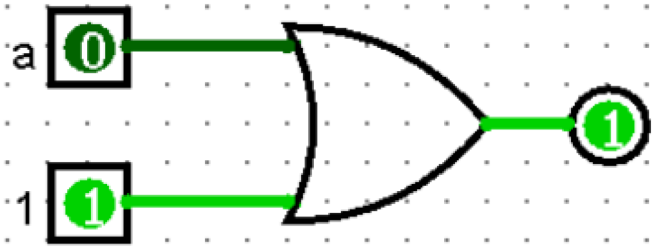
Double Negation

→ (a′)′ = a

Annihilator

- Annihilator for AND:
  → a*0 = 0
- Annihilator for OR:
  → a+1 = 1





Boolean Algebra

- Making sure equation is equal to: F = x' + (y * z)

F = x'*y'*z' + x'*y'*z + x'*y*z' + x'*y*z + x*y*z

F = x'*(y'*z' + y'*z + y*z' + y*z) + x*y*z *(distributive property)*

F = x'*(y'*(z' + z) + y*(z' + z)) + x*y*z *(distributive property)*

F = x'*(y'*(1) + y*(1)) + x*y*z *(complement property)*

F = x'*(y' + y) + x*y*z *(identity property)*

F = x'*(1) + x*y*z *(complement property)*

F = x' + x*y*z *(identity property)*

F = (x' + x) * (x' + y) * (x' + z) (distributivity property)

F = (1) * (x' + y) * (x' + z) *(complement property)*

F = (x' + y) * (x' + z) *(identity property)*

F = (x' + y) * x' + (x' + y) * z *(distributivity property)*

F = x' + (x' + y) * z *(absorption property)*

F = x' + x'*z + y*z *(distributivity property)*

F = x' + (y * z) *(absorption property)*

13 Steps!

Try to do it in fewer steps

Boolean Algebra

- Making sure equation is equal to: F = x’ + (y * z)

Steps	F = x'*y'*z' + x'*y'*z + x'*y*z' + x'*y*z + x*y*z	Original
1	F = x'*(y'*(z'+z) + y*(z'+z)) + x*y*z	Distributive
2	F = x'*(1) + x*y*z	Compliment
3	F = x' + x*y*z	Identity
4	F = (x' + x) * (x' + y) * (x' + z)	Distributive
5	F = 1 * (x' + y) * (x' + z)	Compliment
6	F = ((x' + y)*(x' + z))	Identity
7	F = x'+(y*z)	Distributive

SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

Topics

- Merging Truth Tables

Truth Table: “Don’t Care” Symbol

- In some situations there will be a row in a truth table where an input (or output) doesn’t matter
  - In these instances it is common to use a *don’t care* symbol
  - Often an “x”, sometimes “-“
    - Best practice: define symbol you use
    - We will use “x”

Truth Table: “Don’t Care” Symbol

- Suppose we want a circuit where the output is 1 if at least one input is 1
- **Question**: Truth table complete? Does it cover all possible input combinations?

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

This row represents the following input combinations for inputs *xyz*:

- 001
- 011
- 101
- 111

Truth Table: “Don’t Care” Symbol

- Find two rows which differ at one input
- Replace this input with *don’t care* symbol in one row
- Remove other row

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

a	b	c	F
0	x	0	0
x	0	1	1
…			

Truth Table: “Don’t Care” Symbol

- Find two rows which differ at one input
- Replace this input with *don’t care* symbol in one row
- Remove other row

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

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

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

SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

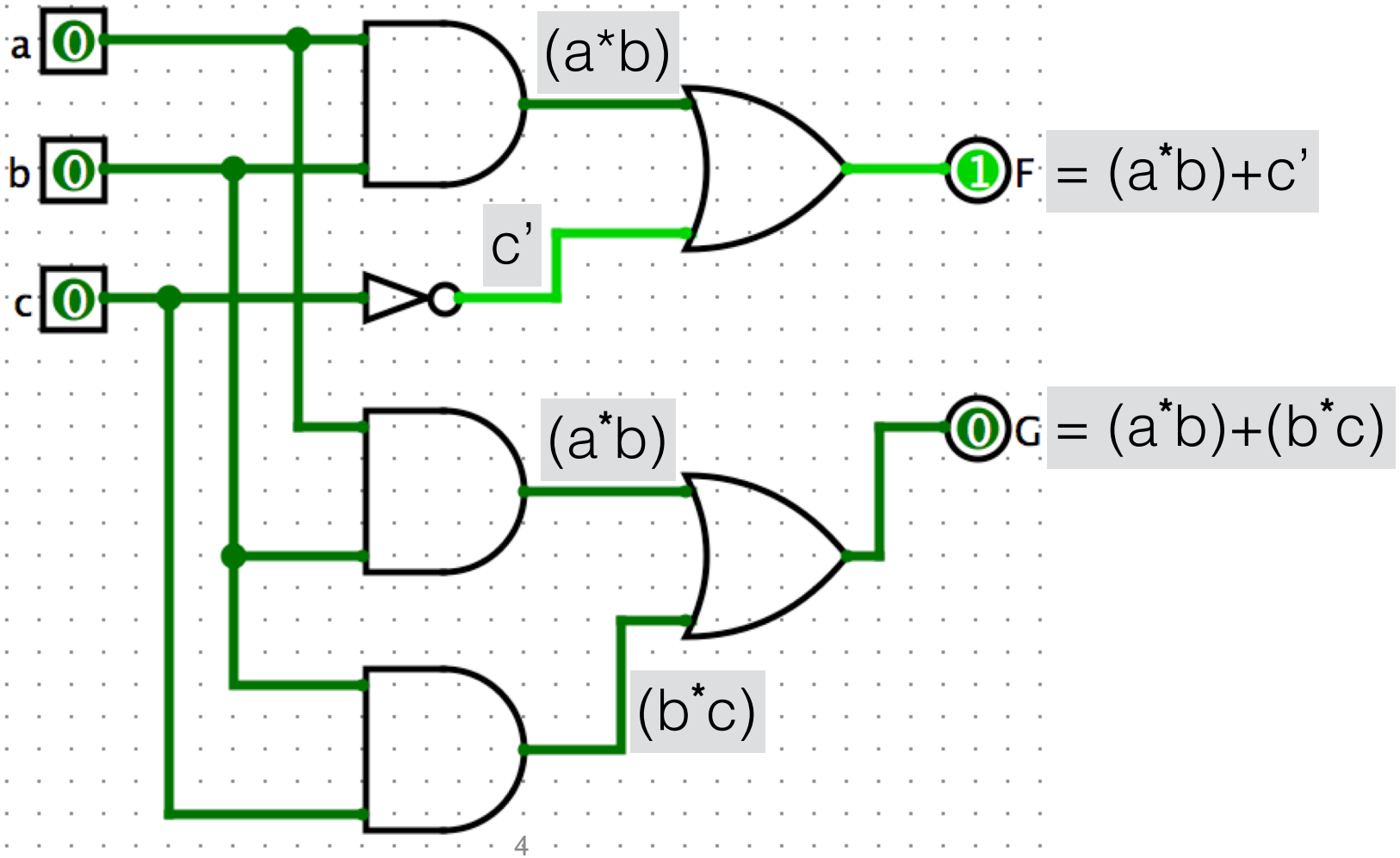
Topics

- Multiple Output Circuits
- Boolean Equations Terminology
- Minterm

Multiple Outputs

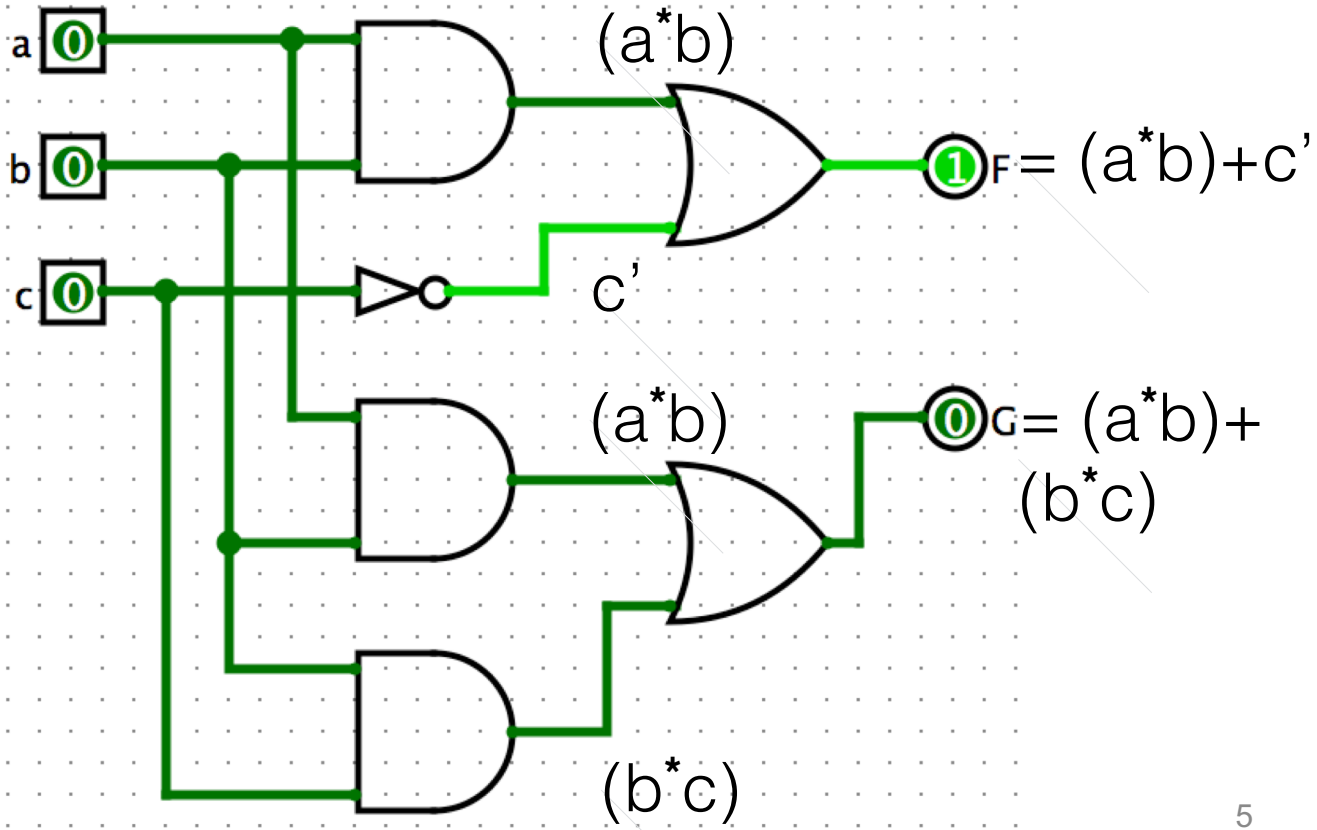
- Circuits can have more that one output
  - Multiple Boolean equations can describe behavior of circuits with multiple outputs
  - Each output can be described with one equation

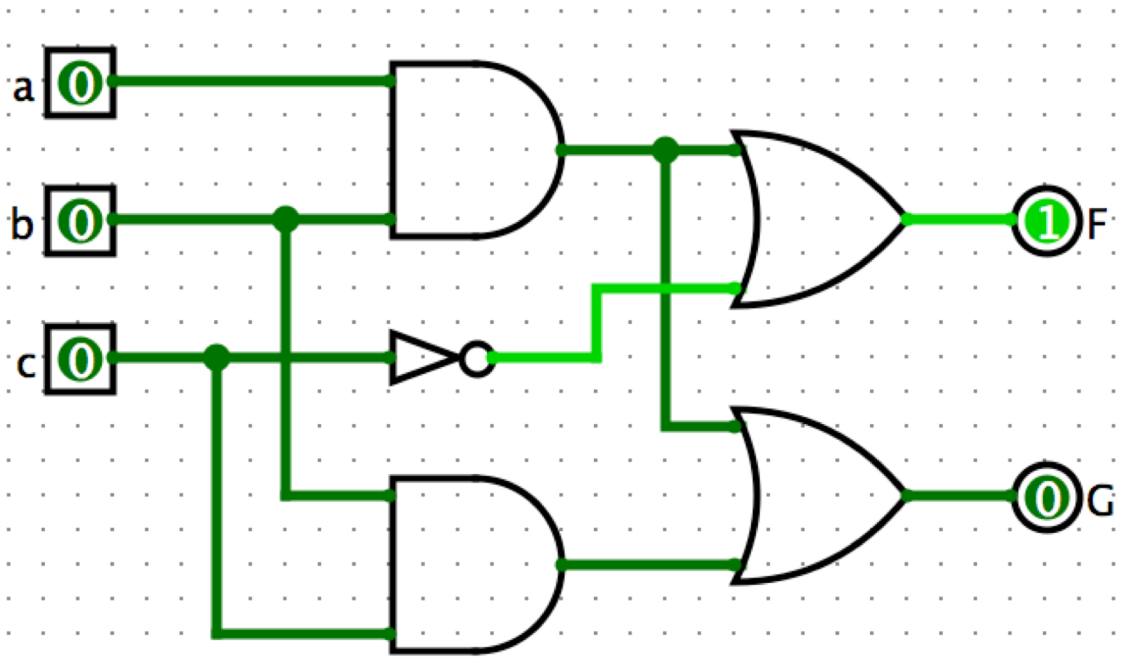
Multiple Outputs



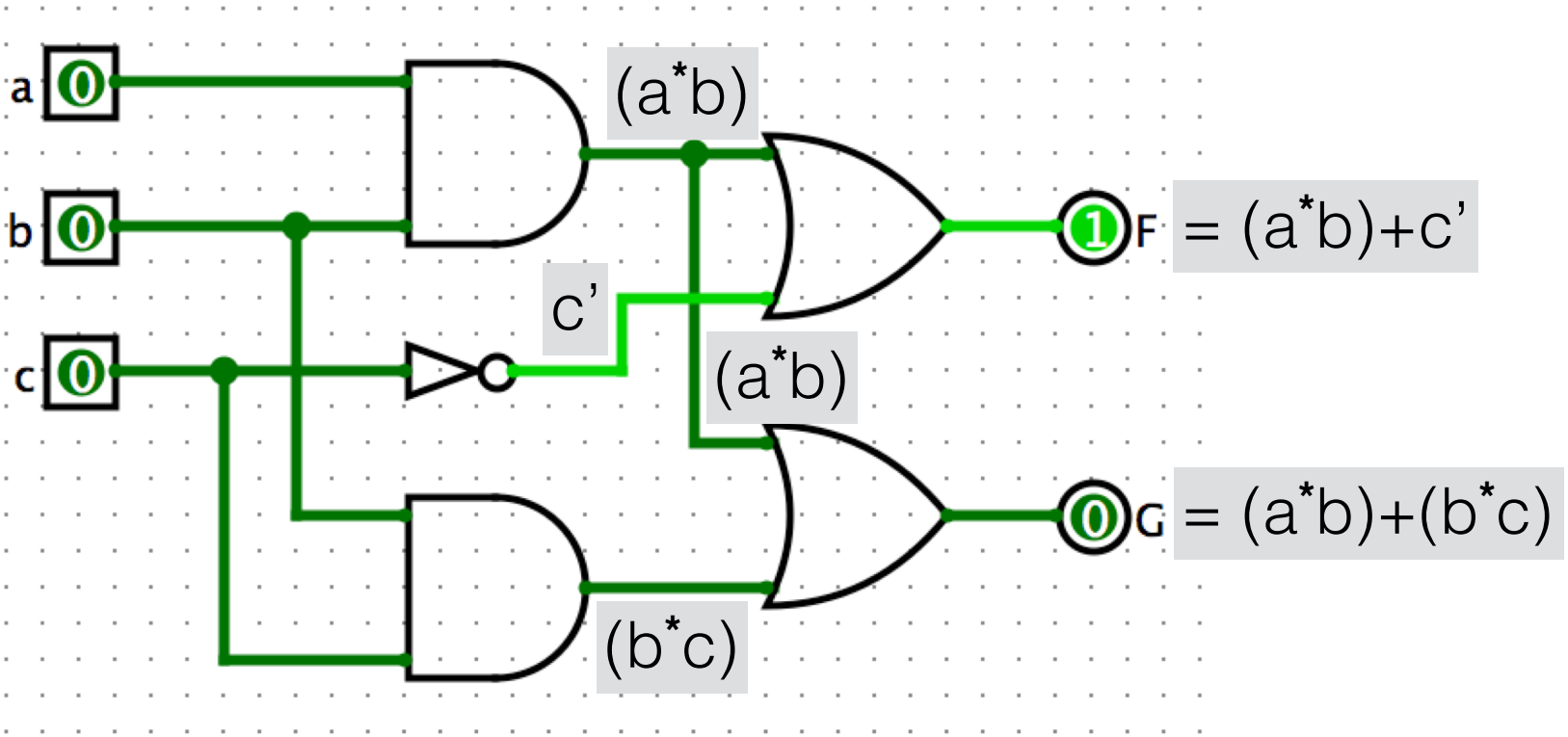
Multiple Outputs

- Do these two circuits have equal behavior?
  - Check if equal: Compare Boolean equations or truth tables





Multiple Outputs



Identical to Boolean equations of first circuit, thus same behavior

Boolean Equations - Terminology

- Example equation: **F(a,b,c) = a'bc + abc' + ab + c**
- ***Variable***
  - Represents a value (0 or 1)
  - Three variables: a, b, and c
- ***Literal***
  - Appearance of a variable, in true or complemented form
  - Nine literals: a', b, c, a, b, c', a, b, and c
- ***Product term***
  - Product of literals (AND gate)
  - Four product terms: a'bc, abc', ab, c
- ***Sum-of-products***
  - Equation written as OR of product terms only
  - Above equation is in sum-of-products form. "F = (a+b)c + d" is not.

Checking Equivalency of Boolean Equations

Sum of Minterms/Canonical Form:

- ***Minterm***: product term with every function literal appearing exactly once, in true or complemented form *(Vahid – Digital Design Slides)*

Checking Equivalency

- Simplify to Sum of Products
- Expand to Sum of Minterms

Example

- Are these boolean equations equivalent?
  F(a,b) = ab’ + a’ **vs** F(a,b) = ab’ + a’b + a’b’

1. ab’ + a’*1	Identity
2. ab’ + a’(b + b’)	Complement
3. ab’ + a’b + a’b’	Distribute

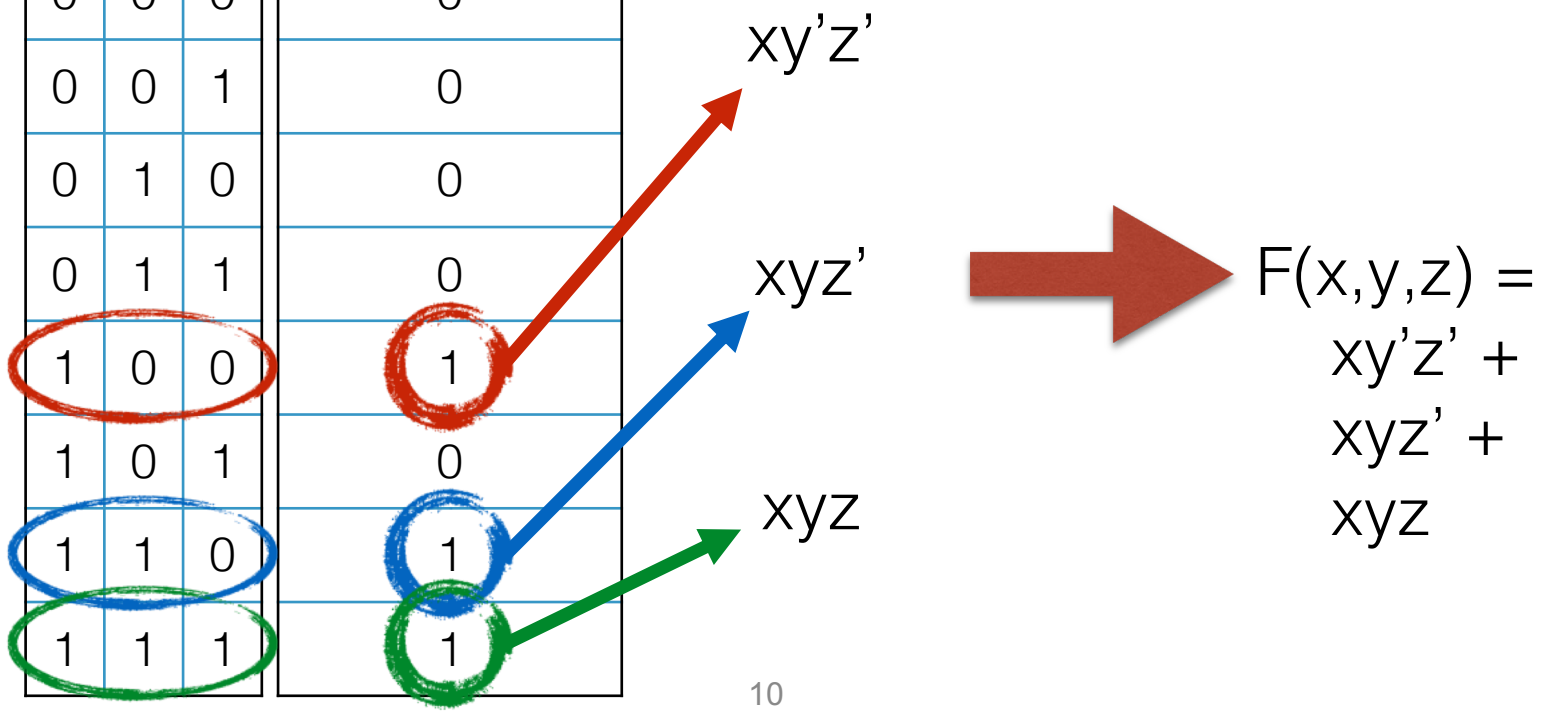
Minterm

- If each of a circuit's input variables are combined with a logic AND in a single term, it is called a ***minterm***
- A *minterm* only includes AND and NOT gates
- If the whole equation consists exclusively of minterms that are combined with logic OR, this form is called ***canonical sum of minterms***

Minterm

- Conversion: Truth Table to Boolean Equation

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



Minterm

- F(x,y,z) = xy'z' + xyz' + xyz
  - → 3 *minterms*
    - Each term includes all inputs/function literal: x, y, z
    - Each term is a product-term (only includes AND and NOT gates)
  - → F is in *canonical sum of minterms* form
    - All *minterms* are combined with logical OR (sum-of-products)

Minterm

- Are the following equations in *canonical sum of minterms* form?

1. F(a,b,c) = abc + a'b'c + a'bc
2. F(a,b,c) = a (bc + bc')

   → F(a,b,c) = abc + abc' (distributive property)
3. F(a,b,c) = a'b'c' + a'b + abc'
4. F(a,b,c) = a
5. F(a,b,c) = ab + bc

Minterm

- Can we turn them into canonical sum of minterm forms?
  3. F(a,b,c) = a'b'c' + **a'b** + abc'
     - **a'b** stands for **a'bc** and **a'bc'** (equal to having a *don't care* for **c** in truth table)
     - Replace non-minterms with all possible minterms

     → F(a,b,c) = a'b'c' + **a'bc** + **a'bc'** + abc'
  4. F(a,b,c) = a

     → F(a,b,c) = abc + abc' + ab'c + ab'c'
  5. F(a,b,c) = ab + bc

     → F(a,b,c) = abc + abc' + a'bc + abc (next: remove redundant terms)

     → F(a,b,c) = abc + abc' + a'bc

SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

Topics

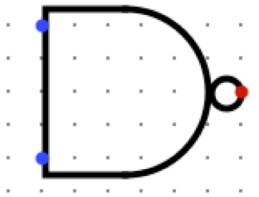
- More Logic Gates & Applications

Logic Gates

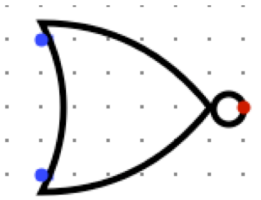
- More than the 3 basic logic gates exist
- Every other gate can be created with those 3 basic gates: AND, OR, NOT

More Logic Gates

- NAND
  - Combination of AND and NOT gate: $(x \cdot y)'$

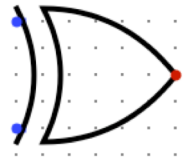


- NOR
  - Combination of OR and NOT gate: $(x + y)'$



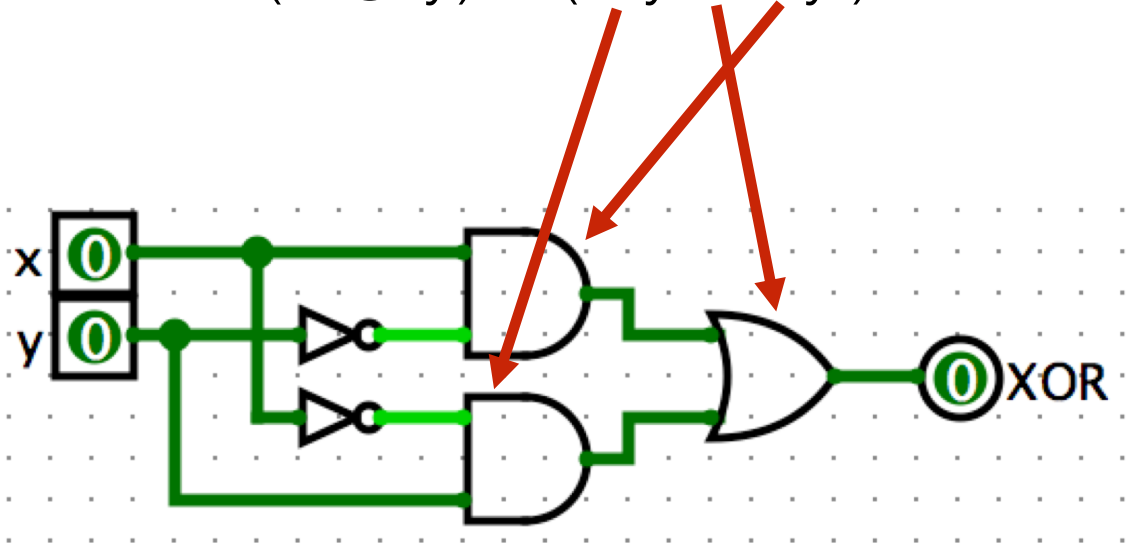
Logic Gates: XOR

- XOR
  - Symbol for boolean algebra: $\oplus$



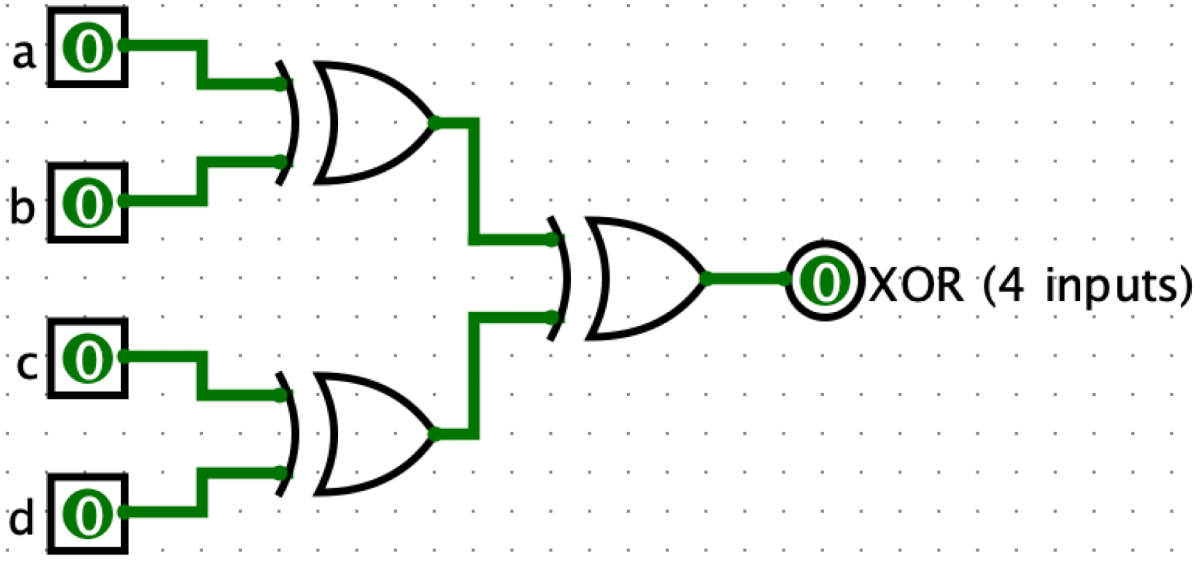
x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$(x \oplus y) = (x' \cdot y + x \cdot y')$



Logic Gates: XOR

- XOR with multiple inputs
  - XOR outputs
    - `1` if there is an odd number of 1s on the inputs
    - `0` if there is an even number of 1s on the inputs



0 XOR 1 =

a	b	c	d	XOR
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Logic Gates: XOR Applied

- A so called *parity bit* indicates if binary value has even or odd amount of 1s
  - XOR generates exactly that information
  - Can be used to:
    - … efficiently create data redundancy to prevent data-loss during hard drive failures
    - … simple check if a communication was erroneous (does not catch all error cases)

Logic Gates: XOR Applied

- Parity HDD combines data from two other drives with *bitwise* XOR:
  - <content HDD 1> **XOR** <content HDD2> **=** <content parity drive>
- Why does that help in case of HDD failures?
  - Loosing one of the three HDDs, two remaining can be used to reconstruct the data of the lost HDD
    - HDD 1 and 2 sizes must be equal
    - Parity HDD must be equal or larger than HDD 1 & 2
  - Parity HDD can secure twice the amount of data than its own size
    - If HDD 1 & 2 each have size of 6 TB, parity HDD secures 12 TB of data with only 6 TB of space

Logic Gates: XOR Applied

- Parity HDD:
  - <content HDD 1> **XOR** <content HDD2> **=** <content parity HDD>
- Failure example:
  1. 01001111 **XOR** 11100001 **=** 10101110
  2. HDD 2 fails and data is lost
  3. 01001111 **XOR** ???????? **=** 10101110
  4. Reconstruct: HDD1 **XOR** parity HDD **=** 11100001

Logic Gates: XOR Applied

- Check if communication was erroneous:
  - Transfer data (e.g. 8 bit) + parity bit of that data
  - Parity bit is XOR of all transferred bits (data: 11001101; parity bit: 1)
  - Receiver calculates parity bit of data and compares to sent parity bit
    - Equal: Communication most likely correct
    - Not equal: Communication was erroneous (e.g. bit “flipped” during transfer)
  - Example: Sent **`11001101 + 1`** but received **`1100110`**<u>**`0`**</u> **`+ 1`**
    - Receiver calculates parity bit of received data: parity bit = 0
    - Received parity bit (1) not equal to calculated parity bit (0) = <u>error in communication!</u>

More Logic Gates

- XNOR
  - Combination of XOR and NOT gate: $(x \oplus y)'$