Today's Expedition…

- TED highlights
- Equivalence Relations
  - Equivalence Classes and Partitions
- Boolean Algebra
  - Boolean functions
  - Sum of Products expansion
  - Logic gates and circuits
- Sections 8.5, 11.1-11.3 in the text

TED 2011 Highlights
(http://conferences.ted.com/TED2011/photos/)

Relations

A binary relation R from set A to set B is a subset of Cartesian product $A \times B$

Example: $A = \text{UW students} \quad B = \text{UW courses}$

$$R = \{(a,b) \mid a \text{ is enrolled in } b\}$$

Example: $A = \{0,1,2\} \qquad B = \{a,b\}$

$$R = \{(0,a),(0,b),(1,a),(2,b)\}$$

Equivalence Relation

A binary relation on a set A is called an *equivalence relation* iff it is reflexive, symmetric, and transitive.

$a \sim b$ a is equivalent to b with respect to a particular equivalence relation

Examples

Equivalence relations

$$R = \{(a,b) \mid a = b\}$$

$$R = \{(a,b) \mid a = b \text{ or } a = -b\}$$

$$R = \{(a,b) \mid a \equiv b (\text{mod } m)\} \text{ where } m \text{ is a positive integer} > 1$$

Not an equivalence relation:

$R = \{(a,b) \mid a \le b\}$ Not symmetric

$R = \{(a,b) \mid b = a + 1\}$ Not reflexive, symmetric, or transitive

Equivalence Classes

- Given an equivalence relation R on set A, the equivalence class of an element a in A is: $[a]_R = \{b \mid (a,b) \in R\}$
- Example:

  $$R = \{(a,b) \mid a \equiv b (\text{mod } 3)\} \text{ over the set of integers}$$

- 3 equivalence classes (congruence classes modulo 3)

  $[0]_3 = \{0, -3, 3, -6, 6, \ldots\}$ All integers with remainder 0

  $[1]_3 = \{1, -2, 4, -5, 7, \ldots\}$ All integers with remainder 1

  $[2]_3 = \{2, -1, 5, -4, 8, \ldots\}$ All integers with remainder 2

Partitions

- Partition of a set S = collection of disjoint nonempty subsets of S whose union is S.
- Theorem: Let R be an equivalence relation on set S. Equivalence classes of R form a partition of S.
  - See Section 8.5 in text for proof.
- Example: The equivalence relation $R = \{(a,b) \mid a \equiv b \pmod 3\}$ over the set of integers results in the following partition of the set of all integers:

$$[0]_3 = \{0, -3, 3, -6, 6, \ldots\}$$
$$[1]_3 = \{1, -2, 4, -5, 7, \ldots\}$$
$$[2]_3 = \{2, -1, 5, -4, 8, \ldots\}$$

Boolean Algebra

Sections 11.1-11.3

Boolean Algebra

- Just like propositional logic
- Variables can take on values 1 or 0
- We will denote the two values as
  0:≡F and **1:≡T**, instead of **False** and **True**.

Boolean Operations

- Correspond to logical NOT, OR, and AND.
- NOT, AND, and OR operators:

$\bar{x} :\equiv \neg x$ $\qquad x \cdot y :\equiv x \wedge y$ $\qquad x + y :\equiv x \vee y$

Precedence order→

Review of Boolean algebra

- NOT is a horizontal bar above the number
  - $\overline{0} = 1$
  - $\overline{1} = 0$
- OR is a plus
  - $0+0 = 0$
  - $0+1 = 1$
  - $1+0 = 1$
  - $1+1 = 1$
- AND is multiplication
  - $0 \cdot 0 = 0$
  - $0 \cdot 1 = 0$
  - $1 \cdot 0 = 0$
  - $1 \cdot 1 = 1$

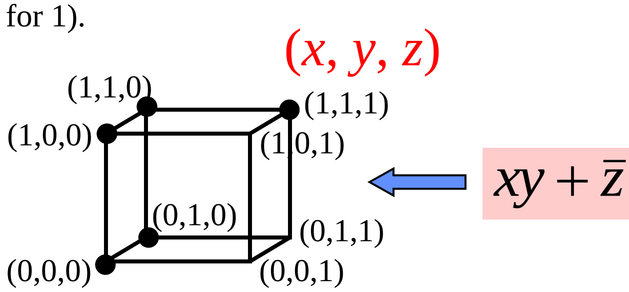
Boolean Expressions and Functions

- Example: Translate $(x+y+z)(\bar{x}\bar{y}\bar{z})$ to a Boolean logic expression
  - $(x \vee y \vee z) \wedge (\neg x \wedge \neg y \wedge \neg z)$
- We can define a Boolean function:
  - $F(x,y) = (x+y)(\bar{x}+\bar{y})$
- And then write a "truth table" for it:

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

N-cube representation of Boolean functions

- Any Boolean function of n variables can be represented by an n-cube with the function values at vertices. (Solid black circle for 1).



Boolean identities

- Double complement:
  $\overline{\overline{x}} = x$
- Idempotent laws:
  $x + x = x, \quad x \cdot x = x$
- Identity laws:
  $x + \mathbf{0} = x, \quad x \cdot \mathbf{1} = x$
- Domination laws:
  $x + \mathbf{1} = \mathbf{1}, \quad x \cdot \mathbf{0} = \mathbf{0}$
- Commutative laws:
  $x + y = y + x, \quad x \cdot y = y \cdot x$
- Associative laws:
  $x + (y + z) = (x + y) + z$
  $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- Distributive laws:
  $x + y{\cdot}z = (x + y){\cdot}(x + z)$
  $x \cdot (y + z) = x{\cdot}y + x{\cdot}z$
- De Morgan's laws:
  $\overline{(x \cdot y)} = \overline{x} + \overline{y}, \quad \overline{(x + y)} = \overline{x} \cdot \overline{y}$
- Absorption laws:
  $x + x{\cdot}y = x, \quad x \cdot (x + y) = x$

also, the Unit Property: $x + \overline{x} = \mathbf{1}$ and Zero Property: $x \cdot \overline{x} = \mathbf{0}$

Sum-of-Products Expansion

- **Theorem:** Any Boolean function can be represented as a sum of products of variables and their complements.
  - **Proof:** By construction from the function's truth table.
- Example: $F(x,y,z) = (x+y)(\bar{x}+\bar{y})$

x	y	$F(x,y)$
1	1	0
1	0	1
0	1	1
0	0	0

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

"minterms"

(x, y, and their complements are called "literals")

Functional Completeness

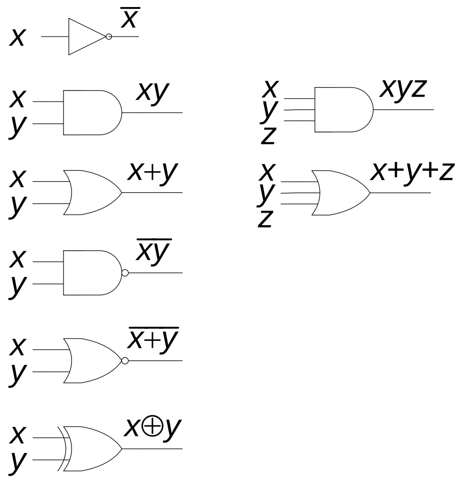
- From previous theorem, any Boolean function can be expressed in terms of $\cdot$, $+$, $\overline{\ }$
  - The set of operators $\{\cdot, +, \overline{\ }\}$ is said to be *functionally complete*.
- Smaller set of functionally complete operators?
  - YES! E.g., Eliminate $+$ using DeMorgan's law. Use

    $$x + y = \overline{\bar{x}\bar{y}}$$

    to write any Boolean function using only $\{\cdot, \overline{\ }\}$.
- NAND $|$ and NOR $\downarrow$ are also functionally complete, each by itself (as a singleton set).
  - *E.g.*, $\neg x = x|x$, and $xy = (x|y)|(x|y)$.

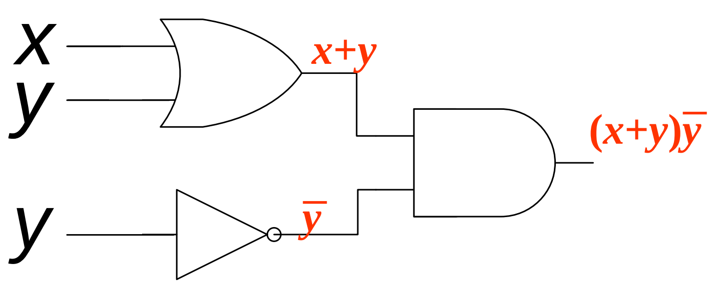
Basic logic gates

- Not
- And
- Or
- Nand
- Nor
- Xor



Boolean Circuits: Example 1

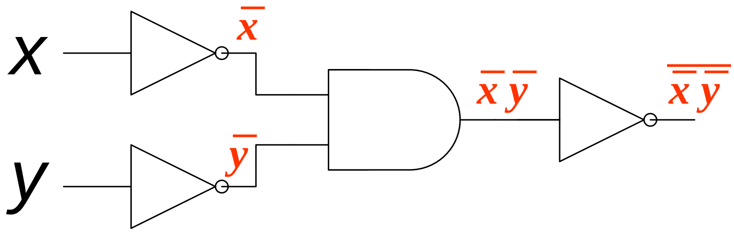
- Find the output of the following circuit



- Answer: $(x+y)\overline{y}$

Example 2

- Find the output of the following circuit

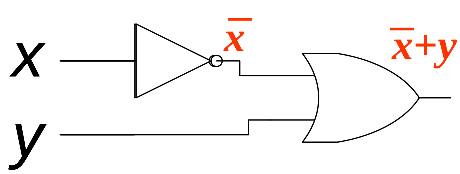


- Answer: $\overline{\overline{x}\,\overline{y}}$

Example 3

- Draw the circuit for the following Boolean function

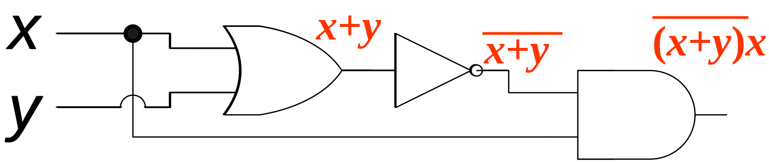
$\overline{x} + y$



Example 4

- Draw the circuit for the following Boolean function

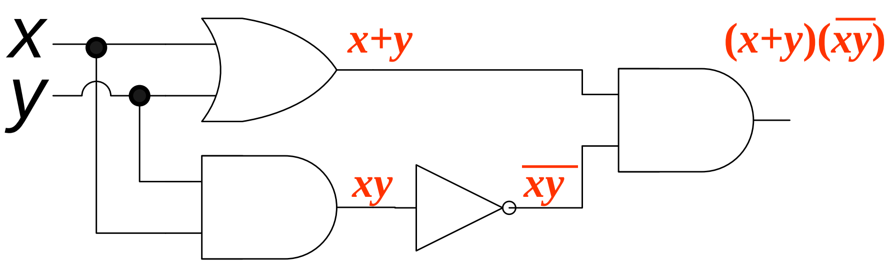
  $\overline{(x+y)}x$



Writing XOR using AND/OR/NOT

- $p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$
- $x \oplus y \equiv (x + y)\overline{(xy)}$

x	*y*	*x*⊕y
1	1	0
1	0	1
0	1	1
0	0	0



How to add binary numbers

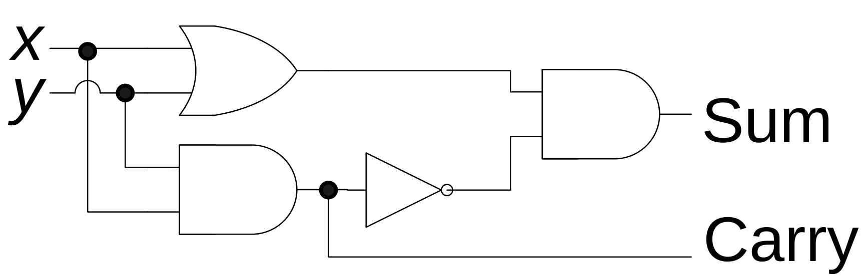
- Consider adding two 1-bit binary numbers *x* and *y*
  - 0+0 = 0
  - 0+1 = 1
  - 1+0 = 1
  - 1+1 = 10

x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Carry is *x* AND *y*
- Sum is *x* XOR *y*
- The circuit to compute this is called a half-adder

The half-adder

- Sum = *x* XOR *y*
- Carry = *x* AND *y*



Using half adders

✦ We can then use a half-adder to compute the sum of two Boolean numbers

$$\begin{array}{rcccc} & 1 & 0 & 0 & \\ & 1 & 1 & 0 & 0 \\ + & 1 & 1 & 1 & 0 \\ \hline & ? & 0 & 1 & 0 \end{array}$$

Full Adder

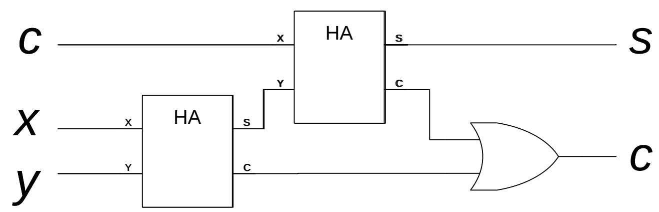
✦ We need to create an adder that can take a carry bit *c* as an additional input
  - ➪ Inputs: *x*, *y*, carry in
  - ➪ Outputs: sum, carry out

✦ This is called a full adder
  - ➪ Will add *x* and *y* with a half-adder
  - ➪ Will add the sum of that to the carry in

x	*y*	*c*	carry	sum
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

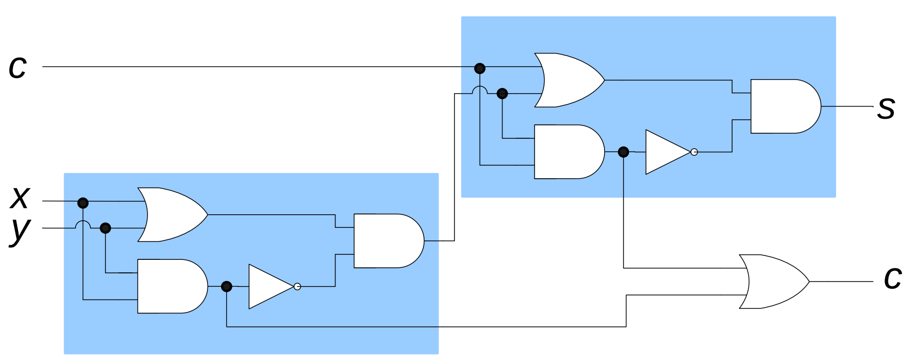
The full adder

- ✦ The "HA" boxes are half-adders



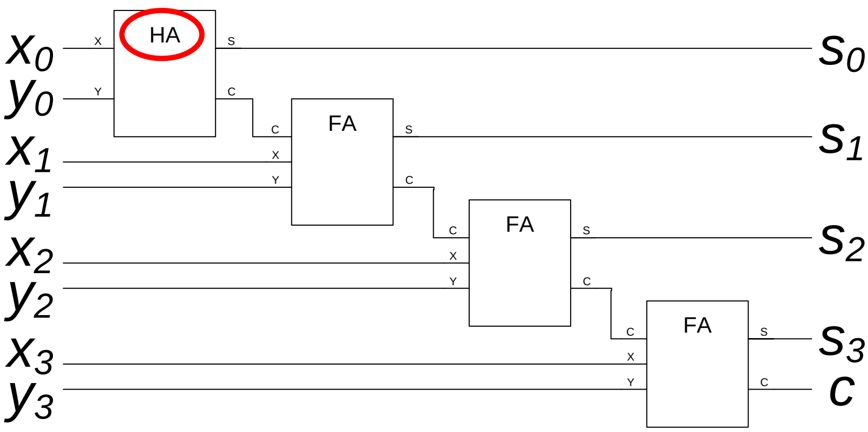
The full adder

- ✦ The full circuitry of the full adder



Adding bigger binary numbers

- Just chain full adders together



Next Class: Graphs and Trees!

Sections 9.1 and 10.1