



Logic

# Why Study Logic?

---

---

- A basis for computer hardware
- A basis for computer programming
- A basis for program optimization
- A basis for specification
- A basis for verification and testing

---

In a certain sense

Computing *is* Logic

---

---

Is all Logic Computing?

No, but  
a lot of it can be reduced to  
computing.

# Flavors of Logic

---

---

- Proposition Logic

- Predicate Logic

- Temporal Logic

- Modal Logics

- Programming Logics

- Fuzzy Logic

} Studied in CS60

} Some exposure in CS80

} Some exposure in CS152  
(Neural Networks)

# Proposition Logic

---

---

- Also known as Switching Logic
- Basic elements are
  - 0 (false)
  - 1 (true)
  - proposition variables (take values 0 or 1)
  - either
    - functions (functional view)
    - connectives (expression view)

# Mostly we use

---

---

- the function view
- and occasionally
  - the expression view

# Proposition Logic Domain

---

---

- {false, true} (for purists)  
or  
{0, 1} (more readable)  
or  
{ $\perp$ ,  $\top$ } (more symmetric)

# Proposition Logic Functions

---

---

- and
- or
- not
- implies
- iff (if, and only if)
- others


# and

---

---

form 1 table:

x	y	and(x, y)
0	0	0
0	1	0
1	0	0
1	1	1

  
arguments

  
results

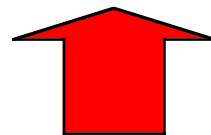
# and

---

---

- form 2 table:

$\text{and}(x, y)$	0	1
0	0	0
1	0	1



results

# and

---

---

- rex "table":
  - $\text{and}(0, 0) \Rightarrow 0;$
  - $\text{and}(0, 1) \Rightarrow 0;$
  - $\text{and}(1, 0) \Rightarrow 0;$
  - $\text{and}(1, 1) \Rightarrow 1;$

# and

---

---

- shorter rex rules (using sequential convention):
  - $\text{and}(1, 1) \Rightarrow 1;$
  - $\text{and}(x, y) \Rightarrow 0;$

# or

---

---

- form 1 table:

x	y	or(x, y)
0	0	0
0	1	1
1	0	1
1	1	1

or

---

---

- form 2 table:

$\text{or}(x, y)$	0	1
0	0	1
1	1	1

# or

---

---

- rex "table"
  - $\text{or}(0, 0) \Rightarrow 0;$
  - $\text{or}(0, 1) \Rightarrow 1;$
  - $\text{or}(1, 0) \Rightarrow 1;$
  - $\text{or}(1, 1) \Rightarrow 1;$

# or

---

---

- shorter rex rules:

- $\text{or}(0, 0) \Rightarrow 0;$

- $\text{or}(x, y) \Rightarrow 1;$

# not

---

---

- form 1 table = form 2 table:

x	not(x)
0	1
1	0

# not

---

- rex rules:
  - $\text{not}(0) \Rightarrow 1;$
  - $\text{not}(1) \Rightarrow 0;$

# implies

---

---

- form 1 table:

x	y	implies(x, y)
0	0	1
0	1	1
1	0	0
1	1	1

# implies

---

---

- form 2 table:

$\text{implies}(x, y)$	0	1
0	1	1
1	0	1

# implies

---

---

- rex rules:
  - $\text{implies}(0, 0) \Rightarrow 1;$
  - $\text{implies}(0, 1) \Rightarrow 1;$
  - $\text{implies}(1, 0) \Rightarrow 0;$
  - $\text{implies}(1, 1) \Rightarrow 1;$

# implies

---

---

- shorter rex rules (sequential):
  - $\text{implies}(1, 0) \Rightarrow 0;$
  - $\text{implies}(x, y) \Rightarrow 1;$

# Concise Summary

(sequential convention applies)

---

---

- $\text{and}(1, 1) \Rightarrow 1;$
- $\text{and}(x, y) \Rightarrow 0;$
- $\text{or}(0, 0) \Rightarrow 0;$
- $\text{or}(x, y) \Rightarrow 1;$
- $\text{not}(0) \Rightarrow 1;$
- $\text{not}(1) \Rightarrow 0;$
- $\text{implies}(1, 0) \Rightarrow 0;$
- $\text{implies}(x, y) \Rightarrow 1;$

# Expression Forms

---

---

- Use for greater readability of certain equalities
- Similar to ordinary discourse

# Expression Forms

---

---

- Function symbols

- and:  $\wedge$       $\cdot$       $\&\&$      *juxtaposition*

Example:  $x \cdot y$  means  $x \wedge y$

- or:  $\vee$       $+$       $\parallel$

- not:  $\neg$       $!$      ' superscript      $\bar{\quad}$  (overbar)

- Example: These mean the same thing:

- $(a \wedge b) \vee (c \wedge \neg d)$

- $ab + cd'$

- Binding order: not, and, or

# Expression Forms

---

---

- Function symbols:
  - implies:  $\rightarrow$   $\supset$
  - iff:  $\equiv$   $=$   $==$

# What We'll Use

---

---

- To start, we'll use

$\wedge$     $\vee$     $\neg$     $\rightarrow$     $\equiv$

- When we discuss circuits, we'll use

$\cdot$     $+$     $'$     $=$

## Logical Equivalences

(These can be shown by substituting all combinations of 0, 1 for variables)

---

---

- $a \wedge b = b \wedge a$
- $a \vee b = b \vee a$
  
- $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
- $(a \vee b) \vee c = a \vee (b \vee c)$
  
- $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$
- $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

## More Logical Equivalences

---

---

- $(a \wedge 0) = 0$

- $(a \wedge 1) = a$

- $(a \vee 0) = a$

- $(a \vee 1) = 1$

## More Logical Equivalences

---

---

●  $\neg(a \wedge b) = (\neg b \vee \neg a)$

●  $\neg(a \vee b) = (\neg b \wedge \neg a)$

} DeMorgan's Laws

●  $(a \vee \neg a \wedge b) = a \vee b$

●  $(a \wedge (\neg a \vee b)) = a \wedge b$

## Logical Equivalences for Implies

---

---

- $(a \rightarrow b) = (\neg a \vee b)$
- $(a \rightarrow b) = \neg(a \wedge \neg b)$
  
- $(0 \rightarrow b) = 1$
- $(1 \rightarrow b) = b$
- $(a \rightarrow 0) = \neg a$
- $(a \rightarrow 1) = 1$

## More Logical Equivalences for Implies

---

---

- $(a \rightarrow bc) = (a \rightarrow b) \wedge (a \rightarrow c)$
- $((a \rightarrow b) \wedge (b \rightarrow c)) \rightarrow (a \rightarrow c)$
- $(a \rightarrow b) = (\neg b \rightarrow \neg a)$

# Checking Relations using the Boole-Shannon Principle

---

---

- Relations hold iff they hold for any **substitution** of 0 and 1 for the variables (**uniformly** throughout the expression)
- Therefore, a relation holds if, choosing *any* variable  $V$ , it holds for  $V = 0$  and for  $V = 1$ .
- But substituting 0 or 1 for a variable often yields **simplifications** that make the relation obvious.

# Example

---

---

- Verify  $(a \rightarrow b) = (\neg b \rightarrow \neg a)$
- Choose  $a$  as the variable.
  - Substituting 0 for  $a$ :
    - $(0 \rightarrow b) = (\neg b \rightarrow \neg 0)$
  - which simplifies to:
    - $1 = (\neg b \rightarrow 1)$ , a known equivalence
  - Substituting 1 for  $a$ :
    - $(1 \rightarrow b) = (\neg b \rightarrow \neg 1)$
  - which simplifies to:
    - $b = (\neg b \rightarrow 0)$

# Boole and Shannon

---

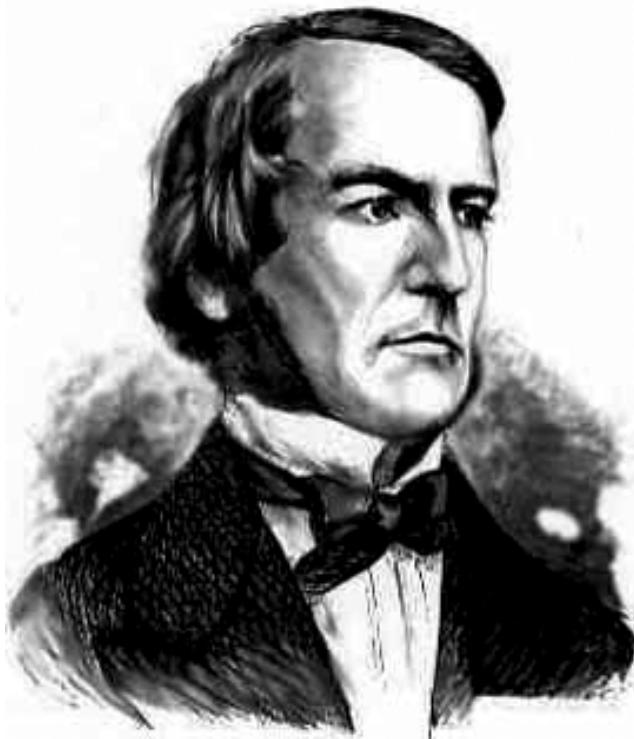
---

- Boole
  - Invented "Boolean algebra" (switching theory)
  - (In modern mathematics, "Boolean algebra" is a more general, abstract, system)
- Shannon
  - Wrote thesis on switching theory
  - Invented "Information theory"
  - Maze-solving mouse
  - Wrote first chess-playing program
  - Wrote paper on the mathematics of juggling

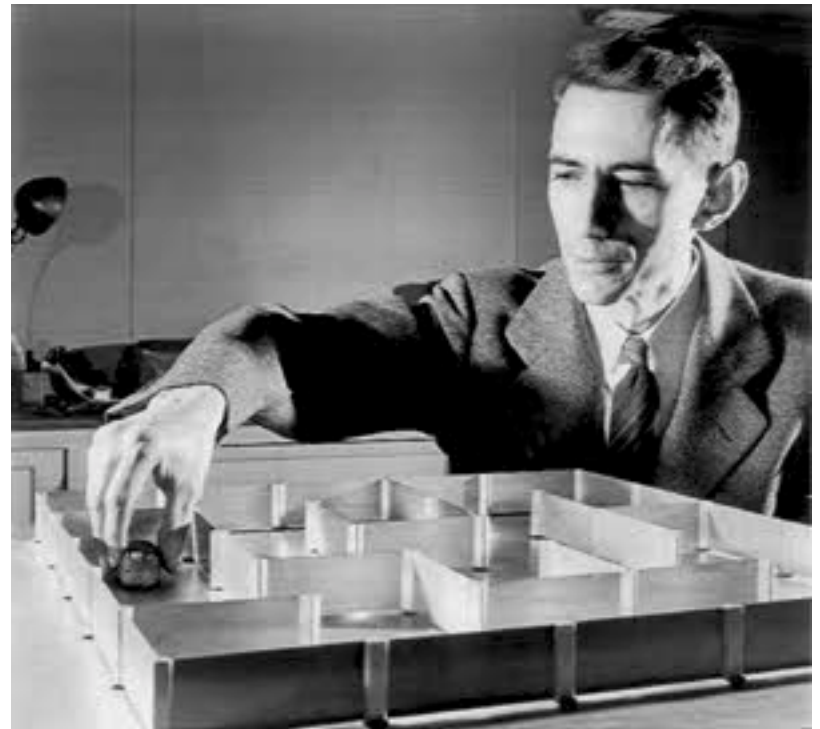
# Boole and Shannon

---

---



George Boole (1815-1864)



Claude Shannon (1916-2001)

# Tautologies

---

---

- An expression that always evaluates to 1 (true) regardless of what value each variable is assigned is called a *tautology*.
- The property of being a tautology can be checked using:
  - Truth-table construction
  - Boole-Shannon Principle, recursively
- Example of a tautology checker (applet):  
<http://www.cs.hmc.edu/~keller/javaExamples/taut/taut.html>