COS 126 Lecture 11: Boolean Logic

Interface between real and abstract machines

Abstract:

everything in machine represented
as binary numbers (sequence of 0-1 values)

Real:

machines constructed of switches, wires
everything is either "on" or "off"

Boolean functions of two variables

Sixteen different functions

```
0 0 1 1    x
0 1 0 1    y
0 0 0 0    constant 0
0 0 0 1    AND (xy)          [decode 00 = 3]
0 0 1 0                      [decode 10 = 2]
0 0 1 1    x
0 1 0 0                      [decode 01 = 1]
0 1 0 1    y
0 1 1 0    XOR (x^y)
0 1 1 1    OR  (x+y)
1 0 0 0    NOR ("not or") [decode 00 = 0]
1 0 0 1    == ("not xor")
1 0 1 0    NOT y (y’)
1 0 1 1
1 1 0 0    NOT x (y’)
1 1 0 1
1 1 1 0    NAND ("not and")
1 1 1 1    constant 1
```

AND, OR and NOT (for example) suffice
to express *any* boolean function

x	y	x’	x’y	y’	xy’	x’y+xy’ = x^y
0	0	1	0	1	0	0
0	1	1	1	0	0	1
1	0	0	0	1	1	1
1	1	0	0	0	0	0

Boolean functions of three variables

256 different functions

0 0 0 0 1 1 1 1	**x**	
0 0 1 1 0 0 1 1	**y**	
0 1 0 1 0 1 0 1	**z**	
0 0 0 0 0 0 0 0	**constant 0**	
0 0 0 0 0 0 0 1	**AND**	**[decode 111 = 7]**
0 0 0 0 1 0 0 0		**[decode 100 = 4]**
0 0 0 1 0 1 1 1	**majority**	
0 1 1 0 1 0 0 1	**odd parity**	
1 0 1 0 1 0 1 0	**NOT z**	
1 1 0 0 1 1 0 0	**NOT y**	
1 1 1 1 0 0 0 0	**NOT x**	
1 1 1 1 1 1 1 0	**NAND**	
1 1 1 1 1 1 1 1	**constant 1**	

65536 different functions of 4 variables

Sum-of-products

Systematic way to express any function

AND, OR, and NOT

form AND terms for each 1 in the function

use v if it corresponds to v = 1

use v' (NOT v) if it corresponds to v = 0

OR the terms together

Ex: majority function

```
x: 0 0 0 0 1 1 1 1
y: 0 0 1 1 0 0 1 1
z: 0 1 0 1 0 1 0 1
m: 0 0 0 1 0 1 1 1
```

$m = x'yz + xy'z + xyz' + xyz$

Example: odd parity function

```
x: 0 0 0 0 1 1 1 1
y: 0 0 1 1 0 0 1 1
z: 0 1 0 1 0 1 0 1
p: 0 1 1 0 1 0 0 1
```

$p = x'y'z + x'yz' + xy'z' + xyz$

Boolean circuits

Circuits are interconnected "GATES"
Values of variables are on the wires

AND:

OR:

NOT:

Use sum-of-products form of function
Example: majority

$$m = x'yz + xy'z + xyz' + xyz$$

Could be done with binary gates
Smaller circuits easy to devise

ANY boolean function can be implemented
with AND, OR, and NOT gates

Building gates

Gate implementation is the basis of the interface between real and abstract machines

Switches, relays, vacuum tubes, transistors,...

Transistor: controlled switch

NOT gate (one transistors)

OR gate (three transistors)

AND gate (four transistors)

ANY boolean function can be implemented
by wiring together transistors

A useful circuit

DECODER

N "inputs"

2^N "outputs"

Turns on precisely one "output"

address is encoded in "inputs"

x y z

x’y’z’ 000

x’y’z 001

x’yz’ 010

x’yz 011

xy’z’ 100

xy’z 101

xyz’ 110

xyz 111

Converts binary to "unary"

Used to select wires by binary ID (address)

Adder circuit

Goal: add two N-bit numbers x and y

Not a bit-by-bit function of the inputs
    because of "carry"

```
1 0 0 1 1 1 1 1
0 0 1 0 0 0 0 1
1 1 0 0 0 0 0 0
```

For each BIT
    three inputs: (x, y, z)
    two outputs: sum, carry

Sum: odd parity circuit

Carry: majority circuit

ADDER: string together one copy of circuit
    for each bit

Timing? [stay tuned]

Adder is a set of switches, wired together