CSEN102 – Introduction to Computer Science

Lecture 9:
Boolean Logic

Prof. Dr. Slim Abdennadher
Dr. Aysha Alsafty
slim.abdennadher@guc.edu.eg,
aysha.alsafty@guc.edu.eg

German University Cairo, Department of Media Engineering and Technology

18.12.2016 - 22.12 2016

Synopsis

Test your understanding...

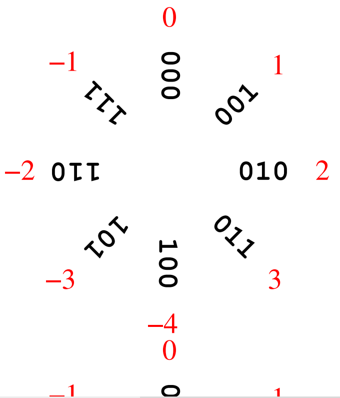
- What is the range of a n-bit unsigned binary integer?
  from 0 to $2^n - 1$
- What is the range of an n-digit unsigned base-b integer?
  from 0 to $b^n - 1$
- What is the range of a n-bit binary two's complement integer?
  from -2^{n-1} to $2^{n-1} - 1$
- What is the advantage of the two's complement representation for negative numenrals over others?
  - Addition of negative numbers is exactly the same a with positive numbers
  - It uses the full capacity of an n-bit numeral
- How do I convert a positive binary into the corresponding two's complement negative?
  - Flip every bit
  - Add 1

Synopsis

As promised:

The idea of the two's complement addition

- For the example, we consider 3-bit two's complement integers
- We add 3 and -2



Synopsis

Test your understanding...

- Define the normalized scientific floating-point notation
  - The numbers are represented as $\pm mantissa \times base^{\pm exponent}$
  - Normalized means that the most significant bit is right of the fraction-point.
- Assume the following encoding for a 16-bit normalized binary floating point:

Sign of mantissa	Mantissa	Sign of exponent	Exponent
1 bit	9 bits	1 bit	5 bits

How do you represent the number 10.01011?

0100101100100010

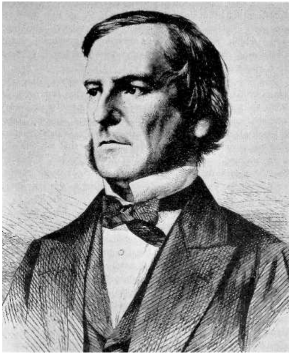
Be prepared to explain why!

New chapter

Boolean Logic

Boolean logic

- Boolean logic is a branch of mathematics that deals with rules for manipulating two logical values true and false.
- Named after George Boole (1815-1864)
- Why is Boolean logic so relevant to computers?
  - Straightforward mapping to binary digits!
  - 0 is false
  - 1 is true



Boolean operations

- Boolean expression is any expression that evaluates to either true or false (ex: $X = 5$, $2 < 4$).
- Boolean expressions are widely used in programming. They are formed from variables and operations.
- Variables are designated by letters (*e. g.*, a, b, c, x, y, $\ldots$). Each variable takes only one of 2 values: 0 or 1.
- The three basic operations are:

AND	product, conjunction of two inputs Expression: xy or $x * y$ or $x \wedge y$
OR	sum, disjunction of two inputs Expression: $x + y$ or $x \vee y$
NOT	negation, complement of one input Expression: x' or $\bar{x}$ or $\neg x$

Truth tables for the basic operations

Truth tables are useful as definition and proof-tool for boolean operators:

AND (conjunction)

x	y	$x * y$
0	0	0
0	1	0
1	0	0
1	1	1

OR (disjunction)

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT (complement)

x	$\neg x$
0	1
1	0

Logical AND

- The AND takes two expressions as input (*e. g.*, A and B)
- It evaluates to TRUE only if both expressions are TRUE
- Written as $A * B$ or AB

Example

- A = It is sunny
- B = I am in vacation
- $(A * B)$ = It is sunny AND I am in vacation

A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR

- The OR takes two expressions as input, (*e. g.*, A and B)
- It evaluates to TRUE if either A is TRUE or B is TRUE or both expressions are TRUE
- Written as $A + B$

Example

- A = It is sunny
- B = I am in vacation
- $(A + B)$ = It is sunny OR I am in vacation OR both

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Logical NOT

- The NOT takes one expression as input, (*e. g.*, A)
- It evaluates to TRUE if A is FALSE (used to invert a meaning)
- Written as A'

Example

- A = It is sunny
- A' = It is not sunny

A	A’
0	1
1	0

Boolean algebra

Boolean expressions are defined through an algebra.

A Boolean Algebra requires:

- A set of values with at least two elements, denoted 0 and 1
- Two binary operations $+$ and $*$
- A unary operation $'$

Boolean algebra

A Boolean Algebra must satisfy these axioms:

$x + 0 = x$	$x * 1 = x$	
$x + 1 = 1$	$x * 0 = 0$	
$x + x = x$	$x * x = x$	
$x + x' = 1$	$x * x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutativity
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributivity
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

Boolean algebra

Note:

- The AND and OR are similar to multiplication and addition.
  - AND yields the same results as multiplication for the values 0 and 1.
  - OR is almost the same as addition, expect for the case 1 + 1.

x	y	$x * y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

- This explains why we borrow the arithmetic symbols $*, +$, 0 and 1 for Boolean operations.
- But there are important differences too.
  - There are a finite number of Boolean values: 0 and 1.
  - OR is not quite the same as addition.
  - NOT is a new operation.

Boolean algebra

A Boolean Algebra must satisfy these axioms:

$x + 0 = x$	$x * 1 = x$	
$x + 1 = 1$	$x * 0 = 0$	
$x + x = x$	$x * x = x$	
$x + x' = 1$	$x * x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutativity
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributivity
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

$x + 0 = x$	$x * 1 = x$	
$x + 1 = 1$	$x * 0 = 0$	
$x + x = x$	$x * x = x$	
$x + x' = 1$	$x * x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutativity
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associativity

Boolean algebra

Think about English examples

- “It is snowing or it is not snowing” is always true ($x + x' = 1$).
- “It is snowing and it is not snowing” can never be true ($x * x' = 0$).
- “I am not not handsome” means that “I am handsome” ($(x')' = x$).

DeMorgan’s laws

These laws explain how to complement arbitrary expressions.

- “I am not rich-or-famous” means that “I am not rich and I am not famous”
- “I am not old-and-bald” means that “I am not old or I am not bald”. But I could be (1) young and bald, or (2) young and hairy or (3) old and hairy.

Boolean expressions

- Using the basic operations, we can form more complex expressions

$$f(x, y, z) = (x + y')z + x'$$

- Terminology and notation
  - f is the name of the function
  - x, y and z are input variables, which range over 0 and 1.
  - A literal is any occurrence of an input or its complement.
- Precedences are important.
  - NOT has the highest precedence, followed by AND, and then OR.
  - Fully parenthesized, the expression above would be written:

$$f(x, y, z) = (((x + (y')) * z) + x')$$

Truth tables

- A truth table represents all possible values of an expression given the possible values of its inputs.
- How do we build a truth table?
  1. Create columns for all variables
  2. Determine the number of rows needed (how many rows should appear?)
     - For n variables, 2^n rows.
  3. Define all possible values for the inputs starting from all 0's to all 1's (*e. g.*, for 3 input variables from 000 to 111)
  4. Find the value of the expression for each input value and fill in the table.

Truth tables – Example

Example

$$f(x, y, z) = (x + y')z + x'$$

x	y	z	$f(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

How to translate a truth table to a boolean expression?

Idea

A	B	Output
0	0	0
0	1	0
1	0	1
1	1	0

- Sum-of-Products-Algorithm:
  - Form AND terms for each row that has 1 as the expected
    - use x if it corresponds to $x = 1$
    - use x' if it corresponds to $x = 0$
  - OR the terms together
- The resulting expression then represents the complete functionality of the truth table.

Sum-of-Products-Algorithm – Example

x	y	z	$f(x,y,z)$	
0	0	0	1	$\leftarrow$
0	0	1	1	$\leftarrow$
0	1	0	1	$\leftarrow$
0	1	1	1	$\leftarrow$
1	0	0	0	
1	0	1	1	$\leftarrow$
1	1	0	0	
1	1	1	1	$\leftarrow$

$$f(x,y,z) \quad = \quad x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz$$

Circuits

Encoding boolean logic in hardware

Expressions and circuits

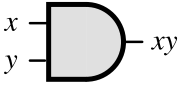
- A circuit is a network of gates that implements one or more boolean functions.
- We can build a circuit for any Boolean expression by connecting primitive logic gates in the correct order.
- Notice that the order of operations is explicit in the circuit.

Primitive logic gates

- A gate is an electronic device that operates on a collection of binary inputs to produce a binary output.
- Each basic operation can be implemented in hardware with a logic gate.

AND (conjunction)

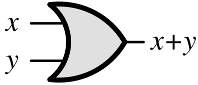
x	y	$x * y$
0	0	0
0	1	0
1	0	0
1	1	1



AND

OR (disjunction)

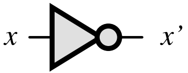
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1



OR

NOT (complement)

x	$\neg x$
0	1
1	0



NOT

Encoding boolean logic in hardware

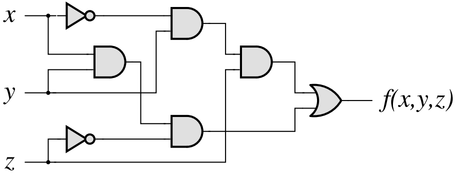
Example

- Truth table:

X	Y	Z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

- Expression:
  $f(x, y, z) =$
  $x' * y * z + x * y * z'$

- Circuit:



Equivalence proof with truth tables

- Two expressions are equivalent iff they always produce the same results for the same inputs

Example ($(x+y)' = x'y'$)

x	y	$x+y$	$(x+y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x	y	x'	y'	$x'y'$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0