Boolean Algebra & Bit Manipulation

Computer Systems Organization

Motivating Example

Design a coffee vending machine

- The machine can dispense coffee, tea, and milk
- It has a button for each choice
- A customer can have at most one of the three choices

Design a program (or circuit) that ensures that at most one of the three choices is selected

Motivating Example

Solution

Use a boolean variable for each button

- C for coffee
- T for tea
- M for milk

Write a function that returns true if either

- (C == true and T == false and M == false)
  or
  (C == false and T == true and M == false)
  or
  (C == false and T == false and M == true)

```
int check_choices(int C, int T, int M)
{
  return (C && !T && !M) ||
         (!C &&  T && !M) ||
         (!C && !T &&  M);
}
```

Boolean Algebra

Integer math

- Operands: integer numbers (0, 1, 2, 3, 4 …)
- Operators: + - * /
- Properties of operators (associativity, commutativity …)
- Examples
  - 1 + 2 = 2 + 1 = 3
  - (3 * 2) * 1 = 3 * (2 * 1) = 6

Boolean algebra: algebraic representation of logic

- Similar to integer math but for logic
- Encode “True” as 1 and “False” as 0

- Operands
  - 0, 1
- Operations
  - “and”: &
  - “or": |
  - “not”: ~

Examples

- 1 & 1 = 1
- 1 | 0 = 1

Claude Shannon introduced boolean algebra to circuits

Boolean Algebra

And

A & B = 1 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or

A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not

~A = 1 when A=0

~	
0	1
1	0

Exclusive-Or (Xor)

A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

Boolean Algebra is Like Integer Math

Commutativity

A | B = B | A

A & B = B & A

A + B = B + A

A * B = B * A

Associativity

(A | B) | C = A | (B | C)

(A & B) & C = A & (B & C)

(A + B) + C = A + (B + C)

(A * B) * C = A * (B * C)

Product distributes over sum

A & (B | C) = (A & B) | (A & C)

A * (B + C) = A * B + B * C

Sum and product identities

A | 0 = A

A & 1 = A

A + 0 = A

A * 1 = A

Zero is product annihilator

A & 0 = 0

A * 0 = 0

Cancellation of negation

~ (~ A) = A

A– (– A) = A

Boolean Algebra is Un-like Integer Math

Boolean: *Sum distributes over product*

A | (B & C) = (A | B) & (A | C) A + (B * C) ≠ (A + B) * (B + C)

Boolean: *Idempotency*

A | A = A A + A ≠ A

"true" or "true" = "true"

"false" or "false" = "false"

A & A = A A * A ≠ A

Boolean: *Absorption*

A | (A & B) = A A + (A * B) ≠ A

"A is true" or "A is true and B is true" = "A is true"

A (1 | (1 & B)) = A

A & (A | B) = A A * (A + B) ≠ A

Boolean: *Laws of Complements*

A | ~A = 1 A + –A ≠ 1

"A is true" or "A is false"

Negation Rules

Negation rules

- ~(A & B) = ~A | ~B
- ~(A | B) = ~A & ~B

Practice

Simplify

C | ~(B & C)

Solution

C | ~(B & C) = C | ~B | ~C

= C | ~C | ~B

= 1 | ~B

= 1

Summary of simplification rules

A | B = B | A

A & B = B & A

(A | B) | C = A | (B | C)

(A & B) & C = A & (B & C)

A & (B | C) = (A & B) | (A & C)

A | (B & C) = (A | B) & (A | C)

A | 0 = A

A & 1 = A

A & 0 = 0

A | A = A

A & A = A

A | ~A = 1

~ (~ A) = A

~(A & B) = ~A | ~B

~(A | B) = ~A & ~B

Practice

Simplify

~(A & B) & (~A | B) & (~B | B)

Solution

~(A & B) & (~A | B) & (~B | B)
= ~(A & B) & (~A | B)
= (~A | ~B) & (~A | B)
= ~A | (~B & B)
= ~A | 0
= ~A

Summary of simplification rules

A | B = B | A
A & B = B & A
(A | B) | C = A | (B | C)
(A & B) & C = A & (B & C)

A & (B | C) = (A & B) | (A & C)
A | (B & C) = (A | B) & (A | C)

A | 0 = A
A & 1 = A
A & 0 = 0
A | A = A
A & A = A

A | ~A = 1
~ (~ A) = A
~(A & B) = ~A | ~B
~(A | B) = ~A & ~B

Other Notations

A & B ———> A.B

———> AB

A | B ———> A + B

~B ———> $\overline{B}$

Bitwise Operations

Bitwise Operations

Boolean operators on bit vectors

Operations applied bitwise

All of the Properties of Boolean Algebra Apply

```
  01101001      01101001      01101001
& 01010101    | 01010101    ^ 01010101    ~ 01010101
__________    __________    __________    __________
  01000001      01111101      00111100      10101010
```

Bitwise Operations in C

Operations &, |, ~, ^ Available in C

- Apply to any "integral" data type
  - `long, int, short, char`
- View arguments as bit vectors
- Arguments applied bit-wise

Examples (Char data type)

- `~0x41 --> 0xBE`
  - $\sim 01000001_2$ --> 10111110_2
- `~0x00 --> 0xFF`
  - $\sim 00000000_2$ --> 11111111_2
- `0x69 & 0x55 --> 0x41`
  - 01101001_2 & 01010101_2 --> 01000001_2
- `0x69 | 0x55 --> 0x7D`
  - 01101001_2 | 01010101_2 --> 01111101_2

Contrast: Logic Operations in C

Contrast to Logical Operators

- `&&, ||, !`
    - View 0 as “False”
    - Anything nonzero as “True”
    - Always return 0 or 1
    - Early termination

Examples (char data type)

- `!0x41   -->   0x00`
- `!0x00   -->   0x01`
- `!!0x41  -->   0x01`

- `0x69 && 0x55   -->   0x01`
- `0x69 || 0x55   -->   0x01`
- `p && *p` (avoids null pointer access)

Shift Operations

Left Shift: `x << y`

- Shift bit-vector `x` left `y` positions
  - Throw away extra bits on left
  - Fill with 0's on right

Argument x	01100010
<< 3	00010000
Log. >> 2	00011000
Arith. >> 2	00011000

Right Shift: `x >> y`

- Shift bit-vector `x` right `y` positions
  - Throw away extra bits on right
  - Left side depends on kind of shift
- Logical shift
  - Fill with 0's on left
- Arithmetic shift
  - Replicate most significant bit on left
  - Useful with two's complement integer representation

Argument x	10100010
<< 3	00010000
Log. >> 2	00101000
Arith. >> 2	11101000

Representing & Manipulating Sets

Representation

- A vector of w bits represents the set A = {0, …, *w*–1}
- if $j \in A$ then set the bit j to 1 in the vector
  - 01101001 { 0, 3, 5, 6 }
  - 76543210

  - 01010101 { 0, 2, 4, 6 }
  - 76543210

Operations

&	Intersection	01000001	{ 0, 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
~	Complement	10101010	{ 1, 3, 5, 7 }

Symmetric difference of A and B is the set of elements that are either in A or in B but not in both

Practice

Show that $A + A.B = A$

Cool Stuff with Xor

- Bitwise xor is form of addition
- With extra property that every value is its own additive inverse
  - A ^ A = 0

```
void funny(int *x, int *y)
{
   *x = *x ^ *y;     /* #1 */
   *y = *x ^ *y;     /* #2 */
   *x = *x ^ *y;     /* #3 */
}
```

	*x	*y
Begin	A	B
1	A^B	B
2	A^B	(A^B)^B = A
3	(A^B)^A = B	A
End	B	A

Main Points

It's All About Bits & Bytes

- Numbers
- Programs
- Text

Different Machines Follow Different Conventions

- Word size
- Byte ordering
- Representations

Boolean Algebra

- Basic form encodes "false" as 0, "true" as 1
- General form like bit-level operations in C
    - Good for representing & manipulating sets