CSE 311 Foundations of Computing I

Lecture 5, Boolean Logic and Predicates

Autumn 2012

Announcements

- Homework 2 available (Due October 10)

Highlights from last lecture

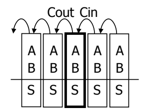
- Boolean algebra to circuit design
- Boolean algebra
  - a set of elements B = {0, 1}
  - binary operations { + , • }
  - and a unary operation { ’ }
  - such that the following axioms hold:

George Boole – 1854

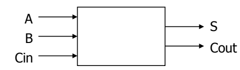
1. the set B contains at least two elements: a, b
2. closure: a + b is in B ; a • b is in B
3. commutativity: a + b = b + a ; a • b = b • a
4. associativity: a + (b + c) = (a + b) + c ; a • (b • c) = (a • b) • c
5. identity: a + 0 = a ; a • 1 = a
6. distributivity: a + (b • c) = (a + b) • (a + c) ; a • (b + c) = (a • b) + (a • c)
7. complementarity: a + a’ = 1 ; a • a’ = 0

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out



A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



S = A’ B’ Cin + A’ B Cin’ + A B’ Cin’ + A B Cin

Cout = A’ B Cin + A B’ Cin + A B Cin’ + A B Cin

Cout = A’ B Cin + A B’ Cin + A B Cin’ + A B Cin

Apply the theorems to simplify expressions

- The theorems of Boolean algebra can simplify expressions
  - e.g., full adder’s carry-out function

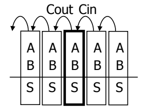
$$
\begin{aligned}
\text{Cout} &= A' B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= A' B\,Cin + A B' Cin + A B\,Cin' + \boxed{A B\,Cin + A B\,Cin} \\
&= A' B\,Cin + A B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= (A' + A) B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= (1) B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A B' Cin + A B\,Cin' + \boxed{A B\,Cin + A B\,Cin} \\
&= B\,Cin + A B' Cin + A B\,Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A (B' + B) Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A (1) Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A\,Cin + A B (Cin' + Cin) \\
&= B\,Cin + A\,Cin + A B (1) \\
&= B\,Cin + A\,Cin + A B
\end{aligned}
$$

adding extra terms creates new factoring opportunities

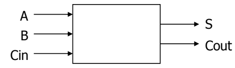
$S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin$

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out



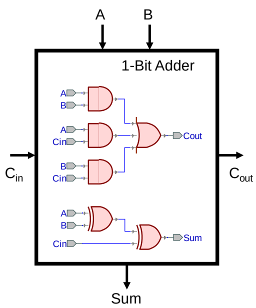
A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

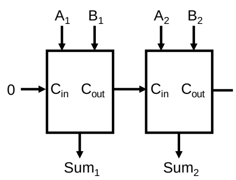


Cout = B Cin + A Cin + A B

S = A xor (B xor Cin)

A 2-bit ripple-carry adder





Mapping truth tables to logic gates

- Given a truth table:
  1. Write the Boolean expression
  2. Minimize the Boolean expression
  3. Draw as gates
  4. Map to available gates

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1

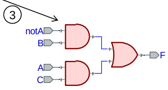
F = A'BC'+A'BC+AB'C+ABC

2

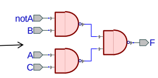
= A'B(C'+C)+AC(B'+B)

= A'B+AC

3



4



Canonical forms

- Truth table is the unique signature of a Boolean function
- The same truth table can have many gate realizations
  - we've seen this already
  - depends on how good we are at Boolean simplification
- Canonical forms
  - standard forms for a Boolean expression
  - we all come up with the same expression

Sum-of-products canonical forms

- Also known as disjunctive normal form
- Also known as minterm expansion

F = *001* *011* *101* *110* *111*

F = A'B'C + A'BC + AB'C + ABC' + ABC

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

F' = A'B'C' + A'BC' + AB'C'

Sum-of-products canonical form (cont)

- Product term (or minterm)
  - ANDed product of literals – input combination for which output is true
  - each variable appears exactly once, true or inverted (but not both)

A	B	C	minterms	
0	0	0	A′B′C′	m0
0	0	1	A′B′C	m1
0	1	0	A′BC′	m2
0	1	1	A′BC	m3
1	0	0	AB′C′	m4
1	0	1	AB′C	m5
1	1	0	ABC′	m6
1	1	1	ABC	m7

short-hand notation for minterms of 3 variables

F in canonical form:
F(A, B, C) $= \Sigma m(1,3,5,6,7)$
= m1 + m3 + m5 + m6 + m7
= A′B′C + A′BC + AB′C + ABC′ + ABC

canonical form ≠ minimal form
F(A, B, C) = A′B′C + A′BC + AB′C + ABC + ABC′
= (A′B′ + A′B + AB′ + AB)C + ABC′
= ((A′ + A)(B′ + B))C + ABC′
= C + ABC′
= ABC′ + C
= AB + C

Product-of-sums canonical form

- Also known as conjunctive normal form
- Also known as maxterm expansion

F = *000* *010* *100*
F = (A + B + C) (A + B′ + C) (A′ + B + C)

A	B	C	F	F′
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

F′ = (A + B + C′) (A + B′ + C′) (A′ + B + C′) (A′ + B′ + C) (A′ + B′ + C′)

Product-of-sums canonical form (cont)

- Sum term (or maxterm)
  - ORed sum of literals – input combination for which output is false
  - each variable appears exactly once, true or inverted (but not both)

A	B	C	maxterms	
0	0	0	A+B+C	M0
0	0	1	A+B+C′	M1
0	1	0	A+B′+C	M2
0	1	1	A+B′+C′	M3
1	0	0	A′+B+C	M4
1	0	1	A′+B+C′	M5
1	1	0	A′+B′+C	M6
1	1	1	A′+B′+C′	M7

short-hand notation for maxterms of 3 variables

F in canonical form:
F(A, B, C) $= \Pi M(0,2,4)$
= M0 • M2 • M4
= (A + B + C) (A + B′ + C) (A′ + B + C)

canonical form ≠ minimal form
F(A, B, C) = (A + B + C) (A + B′ + C) (A′ + B + C)
= (A + B + C) (A + B′ + C)
(A + B + C) (A′ + B + C)
= (A + C) (B + C)

Predicate Calculus

- *Predicate* or *Propositional Function*
  - A function that returns a truth value
- "*x* is a cat"
- "*x* is prime"
- "student *x* has taken course *y*"
- "$x > y$"
- "$x + y = z$"

Quantifiers

- $\forall x\ P(x)$: $P(x)$ is true for every x in the domain
- $\exists x\ P(x)$: There is an x in the domain for which $P(x)$ is true

Statements with quantifiers

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
Equal(x,y)

- $\exists x$ Even(x)
- $\forall x$ Odd(x)
- $\forall x$ (Even(x) $\vee$ Odd(x))
- $\exists x$ (Even(x) $\wedge$ Odd(x))
- $\forall x$ Greater(x+1, x)
- $\exists x$ (Even(x) $\wedge$ Prime(x))

Statements with quantifiers

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
Equal(x,y)

- $\forall x \exists y$ Greater (y, x)
- $\forall x \exists y$ Greater (x, y)
- $\forall x \exists y$ (Greater(y, x) $\wedge$ Prime(y))
- $\forall x$ (Prime(x) $\rightarrow$ (Equal(x, 2) $\vee$ Odd(x))
- $\exists x \exists y$(Equal(x, y + 2) $\wedge$ Prime(x) $\wedge$ Prime(y))

Statements with quantifiers

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
Equal(x,y)

- "There is an odd prime"
- "If x is greater than two, x is not an even prime"
- $\forall x \forall y \forall z$ ((Equal(z, x+y) $\wedge$ Odd(x) $\wedge$ Odd(y))$\rightarrow$ Even(z))
- "There exists an odd integer that is the sum of two primes"

Goldbach's Conjecture

- Every even integer greater than two can be expressed as the sum of two primes

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
Equal(x,y)

Domain:
Positive Integers