CSE 311: Foundations of Computing

Fall 2013

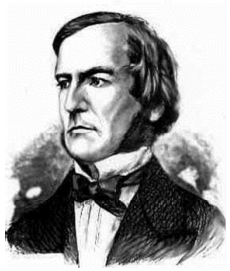
Lecture 5: Canonical forms, predicate Logic

announcements

- Reading assignment
  - Predicates and Quantifiers
    - 1.4, 1.5 7th Edition
    - 1.3, 1.4 6th Edition

review: boolean algebra

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



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

review: 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

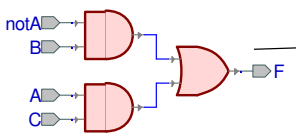
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

① F = A'BC'+A'BC+AB'C+ABC

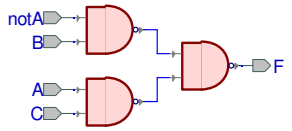
② = A'B(C'+C)+AC(B'+B)

= A'B+AC

③



④



review: 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 form

- also known as Disjunctive Normal Form (DNF)
- 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

sum-of-products canonical form

Product term (or minterm)

- – AND-ed 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) = Σm(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 (CNF)
- 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

s-o-p, p-o-s, and de Morgan's theorem

Complement of function in sum-of-products form:

– F′ = A′B′C′ + A′BC′ + AB′C′

Complement again and apply de Morgan's and get the product-of-sums form:

– (F′)′ = (A′B′C′ + A′BC′ + AB′C′)′

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

product-of-sums canonical form

Sum term (or maxterm)

– **OR-ed 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) = ΠM(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 logic

- **propositional logic**
  - allows us to analyze complex propositions in terms of their simpler constituent parts joined by connectives
- **predicate logic**
  - lets us analyze them at a deeper level…

predicate logic

predicate or propositional function

– a function that returns a truth value, e.g.,

"x is a cat"

"x is prime"

"student x has taken course y"

"x > y"

"x + y = z" or Sum(x, y, z)

Predicates will have variables or constants as arguments.

quantifiers

$\forall$ x P(x)

P(x) **is true for every** x **in the domain**

read as "for all x, P of x"

$\exists$ x P(x)

There is an x **in the domain for which** P(x) **is true**

read as "there exists x, P of x"

statements with quantifiers

- $\exists$ 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))

Domain:
Positive Integers

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

statements with quantifiers

- $\forall$ 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(Sum(x, 2, y) $\wedge$ Prime(x) $\wedge$ Prime(y))

Domain:
Positive Integers

Even(*x*)
Odd(*x*)
Prime(*x*)
Greater(*x*,*y*)
Equal(*x*,*y*)
Sum(x,y,z)

statements with quantifiers

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

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
Sum(x,y,z)

english to predicate logic

- "Red cats like tofu"

Cat(x)
Red(x)
LikesTofu(x)

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

scope of quantifiers

example: Notlargest(x) $\equiv \exists$ y Greater (y, x)
$\equiv \exists$ z Greater (z, x)

truth value:

doesn't depend on y or z "**bound** variables"

does depend on x "**free** variable"

quantifiers only act on free variables of the formula they quantify

$\forall$ x ($\exists$ y (P(x,y) $\rightarrow \forall$ x Q(y, x)))

scope of quantifiers

$\exists x \ (P(x) \wedge Q(x))$ **vs.** $\exists x \, P(x) \wedge \exists x \, Q(x)$

nested quantifiers

- bound variable names don't matter

  $\forall x \exists y P(x, y) \equiv \forall a \exists b P(a, b)$

- positions of quantifiers can sometimes change

  $\forall x (Q(x) \wedge \exists y P(x, y)) \equiv \forall x \exists y (Q(x) \wedge P(x, y))$

- but: order is important...

quantification with two variables

expression	when true	when false
$\forall x \forall y P(x, y)$		
$\exists x \exists y P(x, y)$		
$\forall x \exists y P(x, y)$		
$\exists y \forall x P(x, y)$		

negations of quantifiers

- not every positive integer is prime
- some positive integer is not prime
- prime numbers do not exist
- every positive integer is not prime

de morgan's laws for quantifiers

$$\neg\forall x \; P(x) \equiv \exists x \neg P(x)$$
$$\neg \exists x \; P(x) \equiv \forall x \neg P(x)$$

de morgan's laws for quantifiers

$$\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$$
$$\neg\ \exists x\ P(x) \equiv \forall x\ \neg P(x)$$

"**There is no largest integer**"

$$\begin{aligned}
&\neg\ \exists\ x\ \forall\ y\ \ (x \geq y)\\
\equiv&\forall\ x\ \neg\ \forall y\ \ (x \geq y)\\
\equiv&\forall\ x\ \ \exists\ y\ \neg\ (x \geq y)\\
\equiv&\forall\ x\ \ \exists\ y\quad (y > x)
\end{aligned}$$

"**For every integer there is a larger integer**"