CSE 311

Foundations of Computing I

Pre-Lecture Problem

Create a Boolean Algebra expression for the following truth table (for the function F):

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

Normal Forms

DNF: "OR of ANDs"

$$(\neg x \wedge \cdots \wedge z) \vee \cdots \vee (a \wedge \cdots \wedge \neg b)$$

CNF: "AND of ORs"

$$(\neg x \vee \cdots \vee z) \wedge \cdots \wedge (a \vee \cdots \vee \neg b)$$

In both of these, negations are "pushed" all the way in and must only appear directly next to a literal.

These forms are useful *computationally* because they are easy to work with (fewer cases, easier to simplify, ...).

Canonical Forms

Given a Truth Table…

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

How can we find normal forms?

If we use the same procedure, then we have a canonical form.

This means we can quickly check equality without relying on Boolean Simplification!

Sum-of-Products Canonical Form

AKA Complete Disjunctive Normal Form (CDNF)

AB

1

$AB(C' + C)$

③ **Add the minterms together**

$F = A'B'C + A'BC + AB'C + \cancel{ABC' + ABC}$

AB

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

① **Read T rows off truth table**

② **Convert to Boolean Algebra**

001	A'B'C
011	A'BC
101	AB'C
110	ABC'
111	ABC

F

Sum-of-Products Canonical Form

- 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	$A'B'C'$
0	0	1	$A'B'C$
0	1	0	$A'BC'$
0	1	1	$A'BC$
1	0	0	$AB'C'$
1	0	1	$AB'C$
1	1	0	ABC'
1	1	1	ABC

F in CDNF:

$$F(A, B, C) = A'B'C + A'BC + AB'C + ABC' + ABC$$

Only this one is "CDNF" or Sum-Of-Products Canonical Form

canonical form ≠ minimal form

$$\begin{aligned} F(A, B, C) &= A'B'C + A'BC + AB'C + ABC + ABC' \\ &= (A'B' + A'B + AB' + AB)C + ABC' \\ &= ((A' + A)(B' + B))C + ABC' \\ &= C + ABC' \\ &= ABC' + C \\ &= AB + C \end{aligned}$$

Both of these are in "DNF"

Product-of-Sums Canonical Form

AKA Canonical Conjunctive Normal Form (CCNF)

4 Multiply the maxterms together

F =

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

1 Read F rows off truth table	2 Negate all bits	3 Convert to Boolean Algebra
000	111	$A+B+C$
010	101	$A+B'+C$
100	011	$A'+B+C$

F

Product-of-Sums Canonical Form

AKA Canonical Conjunctive Normal Form (CCNF)

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

1 Read F rows off truth table

2 Negate all bits

3 Convert to Boolean Algebra

000 → 111 → A + B + C

010 → 101 → A + B′ + C

100 → 011 → A′ + B + C

F

4 Multiply the maxterms together

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

Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know (F')' = F
- We know how to get a **DNF** expansion for F'

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

$$(F')' = (A'B'C' + A'BC' + AB'C')'$$

$$F =$$

Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know (F’)’ = F
- We know how to get a **DNF** expansion for F’

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

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

Taking the complement of both sides...

$$(F')' = (A'B'C' + A'BC' + AB'C')'$$

And using DeMorgan/Comp....

$$F = (A'B'C')' \ (A'BC')' \ (AB'C')'$$

$$F = (A + B + C)(A + B' + C)(A' + B + C)$$

Some Administrivia

HW 1 Feedback Released

CSE 311 A

Spring 2017

Home

View Feedback

Submit Homework

To see feedback, go to Canvas and click "View Feedback".

If prompted, log in with your UWNetID—not your CSENetID!

Then, click on the buttons to see your feedback!

Problem Name	Score	View Feedback
HW1		
0	x / 24	HW1-0
1	x / 16	HW1-1
2	x /20	Submitted Online
3	x / 20	Submitted Online
4	x / 10	HW1-4
5	x / 10	HW1-5

Some Administrivia

- Workshops start today!!!!!!!
- Every Wednesday, from 4pm – 6pm in OUG 136, TAs & I will be there to help you work on extra problems.
- We will have whiteboards, markers, and extra problems.
- You can show up to as little or as much of a workshop as you like. We recommend at least 20 minutes though.

Some Administrivia

Group Maker is now online:

If you would like us to help you find a group, go to:

https://grinch.cs.washington.edu/groups

You will be asked some questions to help facilitate finding you a group.

We will re-make groups as necessary **every Friday at 8am.** So, if you want a group this week, make sure to sign up by then.

CSE 311: Foundations of Computing

Lecture 6: Predicate Logic

Predicate Logic

- **Propositional Logic**
  - If the tortoise walks at a rate of one node per step, and the hare walks at a rate of two nodes per step, ...
- **Predicate Logic**
  - If the tortoise is on node x, and the hare is on node 2x, then ...

Predicate Logic

- Propositional Logic
  - Break down a statement into pieces
- Predicate Logic
  - Relates pieces of a statement to each other

What is a "Predicate"?

A predicate is a ***method (function)*** with arguments that returns a ***boolean***.

Examples:

- isPrime(x)
- isLessThan(x, y)
- hasSumOf(x, y, z) {
  return x + y == z;
  }

We will not give "implementations" of predicates. Instead, we'll assumed they're already defined "the way we want".

Defining a Predicate

Cat(x) ::= “x is a cat”

Prime(x) ::= “x is prime”

HasTaken(x, y) ::= “student x has taken course y”

LessThan(x, y) ::= “x > y”

Sum(x, y, z) ::= “x + y = z”

GreaterThan5(x) ::= “x > 5”

HasNChars(s, n) ::= “string s has length n”

Notice that predicates can have varying numbers of arguments and input types.

Domain of Discourse

For ease of use, we define one "type"/"domain" that we work over. This set of objects is called the "**domain of discourse**".

For each of the following, what might the domain be?

(1) "x is a cat", "x barks", "x ruined my couch"

animals "dogs"

(2) "x is prime", "x = 0", "x < 0", "x is a power of two"

(3) "student x has taken course y" "x is a pre-req for z"

Domain of Discourse

For ease of use, we define one "type"/"domain" that we work over. This set of objects is called the "**domain of discourse**".

For each of the following, what might the domain be?

(1) "x is a cat", "x barks", "x ruined my couch"

"mammals" or "sentient beings" or "cats and dogs" or ...

(2) "x is prime", "x = 0", "x < 0", "x is a power of two"

"numbers" or "integers" or "integers greater than 5" or ...

(3) "student x has taken course y" "x is a pre-req for z"

"students and courses" or "university entities" or ...

A Quick Note on “Variable Definition”

What’s wrong here?

isEven(x) ::= “y is even”

A Quick Note on "Variable Definition"

What's wrong here?

isEven(x) ::= "**y** is even"

The definition doesn't make sense, because y isn't defined. It's like writing the following code:

```
isEven(x) { return y % 2 == 0; }
```

Lessons:

- **Be very careful with using "undefined variables"**
- **We need some way of introducing new variables…**

Quantifiers

We use **quantifiers** to talk about collections of objects.

Universal Quantifier ("for all"): $\forall x\, P(x)$

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

read as "**for all x, P of x**"

Examples:

- $\forall x$ Odd(x)
- $\forall x$ LessThan5(x)

Quantifiers

We use quantifiers to talk about collections of objects.

Universal Quantifier ("for all"): $\forall x\, P(x)$

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

read as "for all x, P of x"

Examples:

Are these true? It depends on the domain. For example:

	{1, 3, -1, -27}	Integers	Odd Integers
• $\forall x$ Odd(x)	True	False	True
• $\forall x$ LessThan5(x)	True	False	False

Universal Quantifier ("forall") (Programmatically)

$\forall x\, P(x)$

```
forallP(x) {
    boolean result = true;
    for (x : DOMAIN) {
        result = result && P(x)
}
    return result;
}
```

Quantifiers

We use **quantifiers** to talk about collections of objects.

Existential Quantifier ("exists"): $\exists x\ P(x)$

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

read as "there exists x, P of x"

Examples:

- $\exists x$ Odd(x)
- $\exists x$ LessThan5(x)

Quantifiers

We use **quantifiers** to talk about collections of objects.

Existential Quantifier ("exists"): $\exists x\ P(x)$

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

read as "**there exists x, P of x**"

Examples:

Are these true? It depends on the domain. For example:

	{1, 3, -1, -27}	Integers	Non-Zero Multiples of 10
$\exists x\ Odd(x)$	True	True	False
$\exists x\ LessThan5(x)$	True	True	False

Existential Quantifier ("exists") (Programmatically)

$\exists x\ P(x)$

```
existsP(x) {
   boolean result = false;
   for (x : DOMAIN) {
      result = result || P(x);
   }
   return result;
}
```

Statements with Quantifiers

Just like with propositional logic, we need to define variables (this time **predicates**) before we do anything else. We must also now define a **domain of discourse** before doing anything else.

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Statements with Quantifiers

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

$\forall x\ \exists y\ \text{Greater}(y, x)$

For all pos. int. x, there exists a pos. int. such that y is greater than x.

$\forall x\ \exists y\ \cancel{\text{Greater}(x, y)}$

Less (y, x)

$\forall x\ \exists y\ (\text{Greater}(y, x) \land \text{Prime}(y))$

$\forall x\ (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \lor \text{Odd}(x)))$

$\exists x\ \exists y\ (\text{Sum}(x, 2, y) \land \text{Prime}(x) \land \text{Prime}(y))$

Statements with Quantifiers (Literal Translations)

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

$\forall x\ \exists y\ \text{Greater}(y, x)$

"For every pos. int. x, there is a pos. int. y, such that y > x."

$\forall x\ \exists y\ \text{Greater}(x, y)$

"For every pos. int. x, there is a pos. int. y, such that x > y."

$\forall x\ \exists y\ (\text{Greater}(y, x) \wedge \text{Prime}(y))$

"For every positive integer x, there is a pos. int. y such that y > x and y is prime."

$\forall x\ (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$

"For each pos. int. x, if x is prime, then x = 2 or x is odd."

$\exists x\ \exists y\ (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$

"There exist positive integers x and y such that x + 2 = y and x and y are prime."

Statements with Quantifiers (Better Translations)

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "$x > y$"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "$x = y$"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "$x + y = z$"

Translate the following statements to English

$\forall x\ \exists y\ \text{Greater}(y, x)$

"There is no greatest integer."

$\forall x\ \exists y\ \text{Greater}(x, y)$

"There is no least integer."

$\forall x\ \exists y\ (\text{Greater}(y, x) \wedge \text{Prime}(y))$

"There is always a prime number greater than any positive integer."

$\forall x\ (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$

"Every prime positive integer is either 2 or odd."

$\exists x\ \exists y\ (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$

"There exist prime positive integers that differ by two."

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
Cat(x) ::= "x is a cat"
Red(x) ::= "x is red"
LikesTofu(x) ::= "x likes tofu"

"Red cats like tofu"

$\forall x\ ((Red(x) \wedge Cat(x)) \rightarrow LikesTofu(x))$

"Some red cats don't like tofu"

$\exists x\ ((Red(x) \wedge Cat(x)) \rightarrow \neg LikesTofu(x))$

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
Cat(x) ::= “x is a cat”
Red(x) ::= “x is red”
LikesTofu(x) ::= “x likes tofu”

When we want to put two predicates together like this, we use an “and”.

“Red cats like tofu”

In a “for all”, if we want to assert a property about a particular object, we use an **implication**.

When there’s no leading phrase, it means “for all”.

In an “exists”, if we want to assert a property about a particular object, we use an **and**.

“Some red cats don’t like tofu”

When we want to put two predicates together like this, we use an “and”.

Some means “exists”.

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
Cat(x) ::= "x is a cat"
Red(x) ::= "x is red"
LikesTofu(x) ::= "x likes tofu"

"Red cats like tofu"

$$\forall x\ ((\mathrm{Red}(x) \land \mathrm{Cat}(x)) \to \mathrm{LikesTofu}(x))$$

"Some red cats don't like tofu"

$$\exists y\ ((\mathrm{Red}(y) \land \mathrm{Cat}(y)) \land \neg \mathrm{LikesTofu}(y))$$

Negations of Quantifiers

Predicate Definitions
PF(x) ::= "x is a purple fruit"

$$\forall x\ PF(x)$$

Imagine our domain is {plum, banana, apple}.
Can you write the statement without any quantifiers?

$$PF(plum) \wedge PF(banana) \wedge PF(apple)$$

What is the negation of that statement?

$$\neg(PF(plum) \wedge PF(banana) \wedge PF(apple))$$
$$\equiv \neg PF(plum) \vee \neg PF(banana) \vee \neg PF(apple)$$

"One of the fruits is not purple"

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

Negations of Quantifiers

Predicate Definitions
PF(x) ::= "x is a purple fruit"

$$\forall x\ PF(x)$$

Imagine our domain is {plum, banana, apple}.

Can you write the statement without any quantifiers?

What is the negation of that statement?

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

$$\boxed{\begin{aligned} \neg \forall x\, P(x) &\equiv \exists x\, \neg P(x) \\ \neg \exists x\, P(x) &\equiv \forall x\, \neg P(x) \end{aligned}}$$

"There is no largest integer"

$$\begin{aligned} \forall x \left(\neg(\forall y (x \geq y))\right) &\equiv \forall x \left(\exists y (\neg(x \geq y))\right) \\ &\equiv \forall x \left(\exists y\, (x < y)\right) \end{aligned}$$

"For every integer there is a larger integer"

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

Bound and Free Variables

Consider the following program:

```
hello(x) {
    return x + y;
}
```

In this program, we say "x" is bound and "y" is free.

Bound and Free Variables

Consider the following program:

```
hello(x) {
    return x + y;
}
```

x is defined here

x is used here

y is never defined ☹

In this program, we say "x" is **bound** and "y" is **free.**

Scope of Quantifiers

It's the same idea with quantifiers.

$$\exists y \ \mathrm{Greater}(y, x)$$

$$\forall x \ \exists y \ \mathrm{Greater}(y, x)$$

We figure out what a formula means "inside-out".

So, variables bind to the inner-most quantifier that tries to "capture" them.

$$\forall \cancel{y} \ \exists y \ \mathrm{Greater}(y, x)$$

This quantifier does nothing!

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

Variable Renaming

$\text{NotLargest1}(x) \equiv \exists y \; \text{Greater}(y, x)$ **vs.** $\text{NotLargest2}(x) \equiv \exists z \; \text{Greater}(z, x)$

```
notLargest1(x) {
    boolean result = false;
    for (y : DOMAIN) {
        result = result || y > x
    }
    return result;
}
```

```
notLargest2(x) {
    boolean result = false;
    for (z : DOMAIN) {
        result = result || z > x
    }
    return result;
}
```

These are the same program!
Variable names are irrelevant!

Scope of Quantifiers

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

Scope of Quantifiers

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

This one asserts P and Q of the *same* x.

This one asserts P and Q of potentially different x's.