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

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

- AKA **Disjunctive Normal Form (DNF)**
- AKA **Minterm Expansion**

3

Add the minterms together

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

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

$$\left.\begin{matrix} A'B'C \\ A'BC \\ AB'C \\ ABC' \\ ABC \end{matrix}\right\} F$$

Sum-of-Products Canonical Form

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′
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 canonical form:

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

canonical form ≠ minimal form

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

Product-of-Sums Canonical Form

- AKA **Conjunctive Normal Form (CNF)**
- AKA **Maxterm Expansion**

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 **minterm** 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….

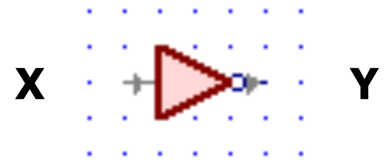
$$F = (A'B'C')' \ (A'BC')' \ (AB'C')'$$

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

Gates Again!

NOT

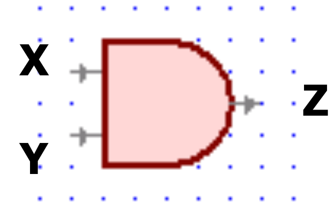
$X' \quad \bar{X} \quad \neg X$



X	Y
0	1
1	0

AND

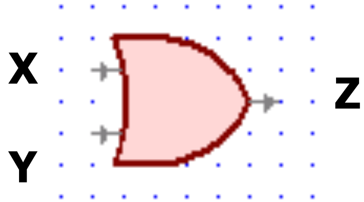
$X \cdot Y \quad XY \quad X \wedge Y$



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

OR

$X + Y \quad X \vee Y$

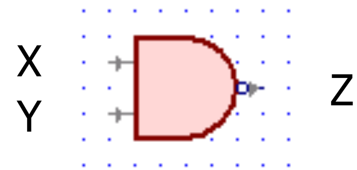


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

More Gates!

NAND

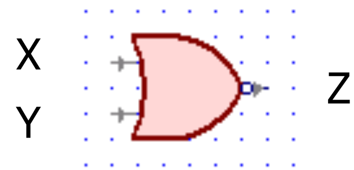
$\neg(X \land Y)$ $(XY)'$



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

NOR

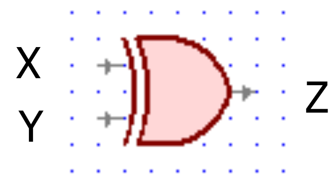
$\neg(X \lor Y)$ $(X+Y)'$



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

XOR

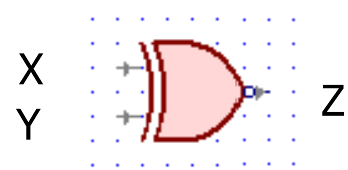
$X \bigoplus Y$



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

XNOR

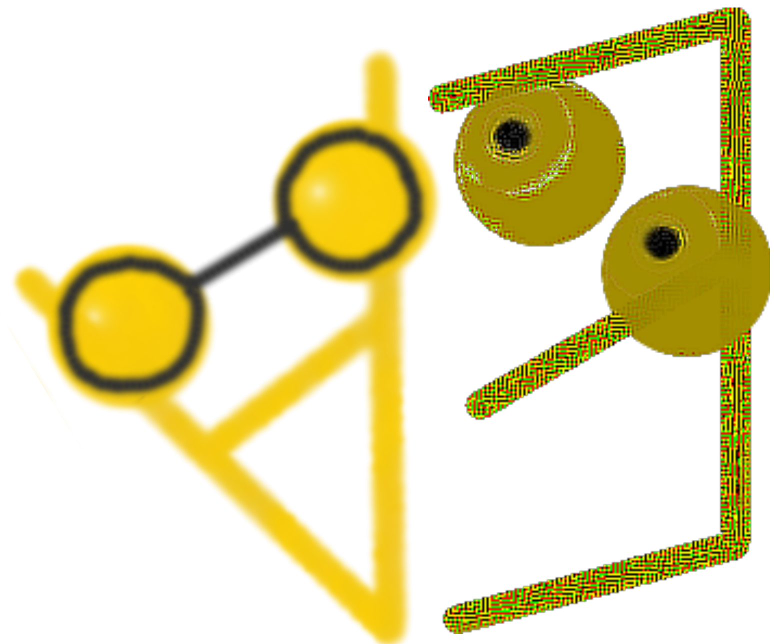
$X \leftrightarrow Y$



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

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)

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"

"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”

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: 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) {
    for (x : DOMAIN) {
        if (!P(x)) {
            return false;
        }
    }
    return true;
}
```

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) {
    for (x : DOMAIN) {
        if (P(x)) {
            return true;
        }
    }
    return false;
}
```

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 (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."

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

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

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

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

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

"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"

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"

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

"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
PurpleFruit(x) ::= “x is a purple fruit”

(*) ∀x PurpleFruit(x) (“All fruits are purple”)

Some possible negations of (*):

(a) “there exists a purple fruit”

(b) “there exists a non-purple fruit”

(c) “all fruits are not purple”

Try your intuition! Which one “feels” right?

Key Idea: In every domain, exactly one of a statement and its negation should be true.

Negations of Quantifiers

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

(*) $\forall x$ PurpleFruit(x) ("All fruits are purple")

Some possible negations of (*):

(a) "there exists a purple fruit"

(b) "there exists a non-purple fruit"

(c) "all fruits are not purple"

Key Idea: In every domain, exactly one of a statement and its negation should be true.

Domain of Discourse
{plum}

(*), (a)

Domain of Discourse
{apple}

(b), (c)

Domain of Discourse
{plum, apple}

(a), (b)

The only choice that ensures exactly one of the statement and its negation is (b).

De Morgan's Laws for Quantifiers

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

"There is no largest integer"

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

"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

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

$\exists x \ (P(x) \land Q(x))$ **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.