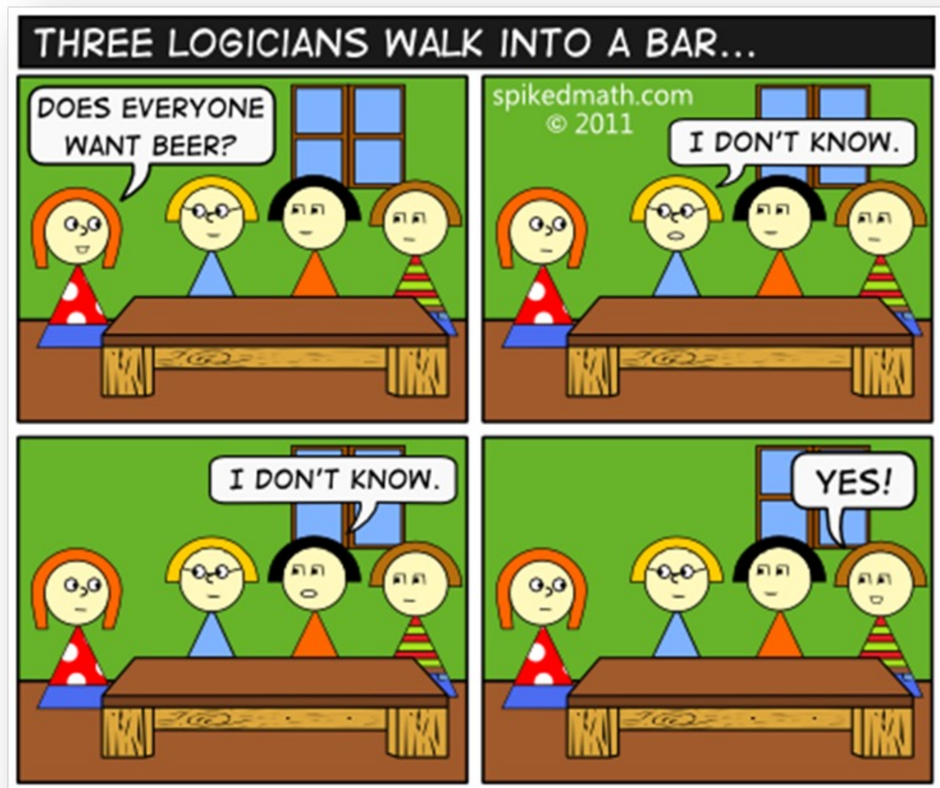
CSE 311: Foundations of Computing

Lecture 5: Predicate Logic



Last class: Canonical Forms

- Truth table is the **unique signature** of a 0/1 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 produce the same expression

Last Time: Sum-of-Products Canonical Form

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

3 Add the minterms together

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

Evaluates to 1 on this row; 0 everywhere else

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

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

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 $\neq$ 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

Evaluates to 0 on this row; 1 everywhere else

A+B+C

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

0

① Read F rows off truth table

② Negate all bits

③ Convert to Boolean Algebra

④ Multiply the maxterms together

000 → 111 → A + B + C

A B C

010 → 101 → A + B’ + C

100 → 011 → A’ + B + C

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

Product-of-Sums Canonical Form

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	$A+B+C$
0	0	1	$A+B+C'$
0	1	0	$A+B'+C$
0	1	1	$A+B'+C'$
1	0	0	$A'+B+C$
1	0	1	$A'+B+C'$
1	1	0	$A'+B'+C$
1	1	1	$A'+B'+C'$

F in canonical form:

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

canonical form ≠ minimal form

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

Predicate Logic

Predicate Logic

- **Propositional Logic**
  - Allows us to analyze complex propositions in terms of their simpler constituent parts (a.k.a. atomic propositions) joined by connectives

- **Predicate Logic**
  - Lets us analyze them at a deeper level by expressing how those propositions depend on the objects they are talking about

"All positive integers x, y, and z satisfy $x^3 + y^3 \neq z^3$."

Predicate Logic

Adds two key notions to propositional logic

- Predicates

- Quantifiers

Predicates

Predicate

- A function that returns a truth value, e.g.,

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”

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 non-empty 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 ...

Quantifiers

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

$\forall x\ P(x)$

$P(x)$ 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

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”

Determine the truth values of each of these statements:

$\exists x\ \mathrm{Even}(x)$ 2 **T** e.g. 2, 4, 6, ...

$\forall x\ \mathrm{Odd}(x)$ 2 **F** e.g. 2, 4, 6, ...

$\forall x\ (\mathrm{Even}(x) \vee \mathrm{Odd}(x))$ **T** every integer is either even or odd

$\exists x\ (\mathrm{Even}(x) \wedge \mathrm{Odd}(x))$ **F** no integer is both even and odd

$\forall x\ \mathrm{Greater}(x+1, x)$ T **T** adding 1 makes a bigger number

$\exists x\ (\mathrm{Even}(x) \wedge \mathrm{Prime}(x))$ **T** Even(2) is true and Prime(2) is true

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

positive ints.

$\forall x\, \exists y$ Greater(y, x) For all x, there exists y, such that

y is greater than x.

For every positive integer x, there is a positive integer y, such that y > x.

$\exists y\, \forall x$ Greater(y, x) there exists y, such that for all x, y gt x

There is a positive integer y such that, for every pos. int. x, we have y > x.

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

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

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

For each positive integer x, if x is prime, then x = 2 or x is odd.

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

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

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

$\forall x\, \exists y$ Greater(y, x)

For every positive integer x, there is a positive integer y, such that y > x.

$\exists y\, \forall x$ Greater(y, x)

There is a positive integer y such that, for every pos. int. x, we have y > x.

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

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

Statements with Quantifiers (Natural 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 positive integer, there is some larger positive integer. (than the first one)

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

There is a positive integer that is larger than every other positive integer.

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

For every positive integer, there is a prime that is larger.

Sound more natural without introducing variable names

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"

"All red cats like tofu"

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

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

"Some red cats don't like tofu"

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

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

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 putting two predicates together like this, we use an "and".

domain restriction

"All Red cats like tofu"

When restricting to a smaller domain in a "for all" we use **implication.**

"Some red cats don't like tofu"

When restricting to a smaller domain in an "exists" we use **and.**

"Some" means "there exists".

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

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

For each positive integer 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 (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

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

Every prime number 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 numbers that differ by two.

Spot the domain restriction patterns

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"

"All Red cats like tofu"

"Red cats like tofu"

When there's no leading quantification, it means "for all".

"Some red cats don't like tofu"

"A red cat doesn't like tofu"

"A" means "there exists".

Statements with Quantifiers (Natural Translations)

Translations often (not always) sound more natural if we

1. Notice "domain restriction" patterns

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

Every prime number is either 2 or odd.

2. Avoid introducing *unnecessary* variable names

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

For every positive integer, there is some larger positive integer.

3. Can sometimes drop "all" or "there is"

$\neg \exists x\ (\text{Even}(x) \wedge \text{Prime}(x) \wedge \text{Greater}(x, 2))$

No even prime is greater than 2.

More English Ambiguity

Implicit quantifiers in English are often **confusing**

Three people that are all friends can form a raiding party $\forall$

Three people I know are all friends with Mark Zuckerberg $\exists$

Formal logic removes this ambiguity

- quantifiers can always be specified
- unquantified variables that are not known constants (e.g, π) are implicitly $\forall$–quantified

Negations of Quantifiers

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

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

What is the negation 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 seems right?

Negations of Quantifiers

Domain = fruits

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

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

What is the negation of (*)?

(a) “there exists a purple fruit”

(b) “there exists a non-purple fruit”

(c) “all fruits are not purple”

Domain of Discourse
{plum, apple}

(*) PurpleFruit(plum) $\wedge$ PurpleFruit(apple)

(a) PurpleFruit(plum) $\vee$ PurpleFruit(apple)

(b) $\neg$ PurpleFruit(plum) $\vee$ $\neg$ PurpleFruit(apple)

$\neg$ PF(plum) $\vee$ $\neg$ PF(apple)

(c) $\neg$ PurpleFruit(plum) $\wedge$ $\neg$ PurpleFruit(apple)

$\exists x . \neg PF(x)$

De Morgan's Laws for Quantifiers

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

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

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 integer larger than every other integer"

$$\neg \exists x\, \forall y\, (x \geq y)$$
$$\equiv \forall x \neg \forall y\, (x \geq y)$$
$$\equiv \forall x\, \exists y \neg (x \geq y)$$
$$\equiv \forall x\, \exists y\, (y > x)$$

"For every integer, there is a larger integer"

De Morgan's Laws for Quantifiers

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

These are **equivalent** but not **equal**

They have different English translations, e.g.:

There is no unicorn $\neg \exists x\ \text{Unicorn}(x)$

Every animal is not a unicorn $\forall x \neg\ \text{Unicorn}(x)$

De Morgan's Laws for Quantifiers

$P \to Q \equiv \neg P \vee Q$

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

"No even prime is greater than 2"

$$\begin{aligned}
&\neg \exists x\, (\mathrm{Even}(x) \wedge \mathrm{Prime}(x) \wedge \mathrm{Greater}(x, 2)) \\
&\equiv \forall x\, \neg(\mathrm{Even}(x) \wedge \mathrm{Prime}(x) \wedge \mathrm{Greater}(x, 2)) \\
&\equiv \forall x\, (\neg(\mathrm{Even}(x) \wedge \mathrm{Prime}(x)) \vee \neg \mathrm{Greater}(x, 2)) \\
&\equiv \forall x\, ((\mathrm{Even}(x) \wedge \mathrm{Prime}(x)) \to \neg \mathrm{Greater}(x, 2)) \\
&\equiv \forall x\, ((\mathrm{Even}(x) \wedge \mathrm{Prime}(x)) \to \mathrm{LessEq}(x, 2))
\end{aligned}$$

"Every even prime is less than or equal to 2."

De Morgan's Laws for Quantifiers

We just saw that

$$\neg \exists x\, (P(x) \land R(x)) \equiv \forall x\, (P(x) \rightarrow \neg R(x))$$

Can similarly show that

$$\neg \forall x\, (P(x) \rightarrow R(x)) \equiv \exists x\, (P(x) \land \neg R(x))$$

De Morgan's Laws respect domain restrictions!
(It leaves them in place and only negates the other parts.)