CMPT-335 Discrete Structures

PREDICATES AND QUANTIFIERS

Truth Verification of those Statements that are not Propositions

- There are no rules in propositional logic that allow to conduct the truth value of the following statements:
- “There is at least one student in this classroom who took three Calculus classes”
- “x+7=10”
- “Each computer in this classroom is connected to the network”
- “y is a student”

Predicates

- Statements like
  - ➢ $x+7=10$
  - ➢ y is a student
  - ➢ $x \geq 3$

have two parts.

- The first part, a variable, is the subject of the statement
- The second part is the predicate. It refers to a property that the subject of the statement may have

Predicates

- Predicate is a dual-purpose term
- On the one hand, a predicate is a property or description of subjects. The following predicates are all shown in red and bold:
  - James **is tall**.
  - The bridge **is long**.
  - 19 **is a prime number**.
- On the other hand, a predicate is also used as a synonym of a propositional function, where the description is related not to a certain subject, but to a variable.

Predicates and Propositional Functions

- If a statement depends on a variable, this statement is called a propositional function (often a propositional function itself is called a predicate)
- We can denote such a statement by $P(x)$.
- Once a value has been assigned to the variable x, the statement $P(x)$ becomes a proposition, which has a truth value.
- $P(x)$ is also referred to as a 1-place predicate or simply a predicate

Multivariable Propositional Functions

- Multivariable propositional functions depend on more than one variable. For example,
  - x is taller than y
  - a is greater than one of b, c
  - x is at least n inches taller than y
  - $x+y=z$
  - x and y are students

n-ary Predicates

- A multivariable propositional function depends on more than one variable. For example, the propositional function x+y=z depends on 3 variables
- Once some values have been assigned to the variables x, y, z, the statement x+y=z becomes a proposition, which has a truth value
- A statement of the form $P\left(x_1, x_2, \ldots, x_n\right)$ is called an n-place predicate or n-ary predicate.

Propositional Functions and Programming

- Propositional functions are widely used in computer programming and algorithm design for branching programs and algorithms

```
If <propositional function(x, y, z)>
then <statement>
else  <statement>
```

- For example:

```
If (((x>0) and (x<y)) or (y>z))
then <statement>
else <statement>
```

Propositional Functions and Programming

- Suppose we need to calculate the values of the following function

$$f(x) = \begin{cases} x^2, & -1 < x \le 10 \\ x^3, & \text{otherwise} \end{cases}$$

  - This is resulted in:

```
If ((x>-1) & (x<=10))
then f(x)=$x^2$
else f(x)=$x^3$
```

Propositional Logic and Artificial Intelligence

- Propositional logic is one of the main and most important tools used in Artificial Intelligence.
- Propositional functions are used to simulate and describe thinking and decision making
- These applications of propositional logic will be considered in detail in the Artificial Intelligence course (CMPT-420/CMPG-720)

Propositional Logic and Correctness of Algorithms

- Propositional logic is widely used to verify the correctness of algorithms and computer programs, particularly to verify the correctness of while loops
- To prove the correctness of algorithm segments or of while loops, preconditions and postconditions are used

Precondition and Postcondition

- Precondition is a statement that describes a valid input of some algorithm or program segment
- Postcondition is a statement that the output of some algorithm or program segment should satisfy when the algorithm (the program segment) has run
- Preconditions and Postconditions are widely used to verify the correctness of algorithms and programs

Precondition and Postcondition

- Preconditions and Postconditions are widely used to verify the correctness of algorithms and programs
- This verification is based on the verification of the following implication:

  Precondition $\rightarrow$ Postcondition

- If Precondition comprehensively describing a valid input is true, then a Postcondition describing a valid output must be true, if a corresponding algorithm is correct
- If Precondition comprehensively describing a valid input is true, but Postcondition describing a valid output is false, then based on the definition of implication, an algorithm is not correct

Precondition and Postcondition and Correctness of Algorithm

Precondition	Postcondition	Precondition→ Postcondition	Correctness of Algorithm
0	0	1	An algorithm is correct at least in terms of not creating a valid output from invalid input
0	1	1	An algorithm may or may not contain a bug
1	0	0	An algorithm is incorrect
1	1	1	An algorithm is correct

Precondition and Postcondition

- Example 1. Let us swap the values of two variables x and y.
- Algorithm: $\boxed{\texttt{temp = x ; x = y; y = temp}}$
- Precondition: “(x==a) & (y==b)”
- Postcondition: “(x==b) & (y==a)”

Precondition and Postcondition

- Example 2. If x <0 or x=10, increment x until it becomes >= 20 (x is a real number).
- Algorithm:

```
double x
if ((x<0) \/ (x==10)) then
while (x<20) {x = x+1}
```

- Precondition: "$(x<0) \lor (x==10)$"
- Postcondition: "$(x>=20)$"

Semantics

- If logical propositions are to have meaning, they need to describe something. Up to now, propositions and propositional functions such as "Johnny is tall", "Debbie is 5 years old", and "$x^2=-1$" had no intrinsic meaning. Either they are true or false, but not more
- In order to make such propositions and propositional functions meaningful, we need to have a **domain** (or universe) of discourse, (or simply **domain** (or universe)) i.e. a collection of subjects about which the propositions relate.
- Question: What are the domains for the three propositions above?

Semantics

- Question: What are the domains for the following propositions and propositional functions?
- "Johnny is tall", "Debbie is 5 years old"

  Possible answers: {Johnny, Debbie}; {People in the world}; {People living in Bronx}, {Students in this classroom}, etc.
- "$x^2=-1$"

  Possible answers: C – the set of complex numbers; R – the set of real numbers, Z – the set of integer numbers
- **Depending on the particular domain**, a proposition can be true or false

Semantics

- Semantics is very important in computer programming
- Every single data structure and even every single variable that are used in a computer program always have their domain of discourse (type).
- For example, a variable x can be declared as double or integer or string or char.
- Depending on this declaration, such expressions as x=3.5, x=5, x='Class', x=-1 can be meaningful and acceptable or not.

Quantifiers

- There are statements, which assert that some property is true for all values of a variable in a particular domain ("Each student in this classroom takes the Discrete Structures course in Spring semester 2019").
- There are also statements, which assert that there is an element in a particular domain with a certain property ("There is at least one person in this classroom who is not a student").
- Such statements can be formulated using **quantifiers**

Quantifiers

There are two quantifiers

- Universal Quantifier

  "∀" reads "for all" or "Each" or "Every"

- Existential Quantifier

  "∃" reads "there exists" or "there is at least one"

➢ A quantifier is placed in front of a propositional function and binds it to obtain a proposition with a semantic value.

Quantifiers

- A statement (propositional function) P(x) is quantified if there is a quantifier in a front of it, which is applied to it: $\forall x\, P(X) \quad \exists x\, P(X)$

- $\forall$x P(x) is TRUE if P(x) is true for every single x.
- $\forall$x P(x) is FALSE if there is an x for which P(x) is false.

- $\exists$x P(x) is TRUE if there is an x for which P(x) is true.

- $\exists$x P(x) is FALSE if P(x) is false for every single x.

Quantifiers

- The truth table for quantifiers

The Truth Table for Quantifiers		
Statement	**When True?**	**When False?**
$\forall x\, P(X)$	**P(x) is true for every x**	**There exists if only one value of x for which P(x) is false**
$\exists x\, P(X)$	**There is an x for which P(x) is true**	**P(x) is false for every x**

Universal Quantifier

- "$\forall x \; P(x)$" is true when every instance of x makes $P(x)$ true when plugged in
- Like conjunctioning over entire domain of $P(x)$

$$\forall x\, P(x) \Leftrightarrow P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \ldots$$

- Example: Each non-negative real number has a square root

Existential Quantifier

- "$\exists x \; P(x)$" is true when an instance can be found which when plugged in for x makes $P(x)$ true
- Like disjunctioning over entire domain of $P(x)$

$$\exists x \, P(x) \quad \Leftrightarrow \quad P(x_1) \vee P(x_2) \vee P(x_3) \vee \ldots$$

- Example: There exist a complex number whose square is a negative real number
- (i^2=-1, i is an imaginary unit)

Preconditions, Postconditions and Quantifiers

- Returning to our two examples of Preconditions/Postconditions used for verification of algorithm correctness (slides 15,16), we can now add quantifiers to our Preconditions/Postconditions

Precondition and Postcondition

- Example 1. Let us swap the values of two variables x and y.
- Algorithm: $\boxed{\text{temp = x ; x = y; y = temp}}$
- Precondition: “(x==a) & (y==b)”
- Postcondition: “(x==b) & (y==a)”

$$\forall x \forall y \left(\left(x == a \right) \wedge \left(y == b \right) \right) \xrightarrow{\text{After all steps of an algorithm}} \left(\left(x == b \right) \wedge \left(y == a \right) \right)$$

Precondition and Postcondition

- Example 2. If x <0 or x=10, increment x until it becomes >= 20 (x is a real number).
- Algorithm:

```
double x
if ((x<0) \/ (x==10)) then
while (x<20) {x = x+1}
y=x // y is used in postcodition
```

-
- Precondition: “(x<0) \/ (x==10)”
- Postcondition: “(y>=20)”

$$\forall x \in R\big((x<0)\vee(x==10)\big)\xrightarrow{\text{After all steps of an algorithm}}\exists y\,(y\geq 20)$$

Multivariate Quantification

- Quantification involving only one variable is fairly straightforward. Just a bunch of OR's or a bunch of AND's.
- When two or more variables are involved each of which is bound by a quantifier, the order of the binding is important and the meaning often requires some thought.

Parsing Example

Question: If the domain for x, y, and z is the natural numbers {0,1,2,3,4,5,6,7,…} what's the truth value of $\exists x \forall y\, \exists z\, P(x, y, z)$; $P(x, y, z) =$ "$y - x \geq z$" ?

Answer: True.

- For any "exists" we need to find a positive instance.
- Since x is the first variable in the expression and is "existential", we need a number that works for all other y, z. Set $x = 0$ (want to ensure that $y - x$ is not too small).
- Now for each y we need to find a positive instance z such that $y - x \geq z$ holds. Plugging in $x = 0$ we need to satisfy $y \geq z$, so set $z = y$.

Parsing Example

- Question: If the domain for for x, y, and z is the natural numbers {0,1,2,3,4,5,6,7,…} , did we have to set $z = y$ to ensure that $\exists x \forall y \exists z\, P(x,y,z)$ where $P(x, y, z) =$ "$y - x \geq z$" is true?
- Answer: No. Could also have used the constant $z = 0$. There are other valid solutions.

Parsing Example

Question: Isn't it simpler to satisfy $\exists x\ \forall y\ \exists z\ (y - x \geq z)$ by setting $x = y$ and $z = 0$?

Answer: No, this is illegal ! The existence of x comes before we know about y. I.e., **the scope of x is higher than the scope of y !**

So we have to find first such x that does not depend on y. Thus, it is illegal to choose x depending on y.

Order matters

- Set the domain of discourse to be all real numbers .
- Let $P(x, y) =$ “$x < y$”.
- Question: What does $\forall x\, \exists y\, P(x,y)$ mean?
- Answer: $\forall x\, \exists y\, P(x,y)$:
- “All numbers x admit a larger number y ”
- Question: What’s the truth value of this expression?
- Answer: True. For any real number there is a larger real number

Order matters

$P(x, y) =$ "$x < y$"

- Question: What does $\exists y \, \forall x \, P(x,y)$ mean?
- Answer: $\exists y \, \forall x \, P(x,y)$:
  "Some number y is larger than all x"
- Question: What's the truth value of this expression?
- Answer: False. There is no the largest real number. Additionally, the number cannot be larger than itself.

Order matters –but not always

- If we have two quantifiers of the same kind, order does not matter.
- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$ because these are both interpreted as "for every combination of *x* and *y*…"
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$ because these are both interpreted as "there is a pair x , y…"

Predicates - the meaning of multiple quantifiers

- $\forall x \forall y\ P(x,y)$ P(x,y) true for all (x, y) pairs.
- $\exists x \exists y\ P(x,y)$ P(x,y) true for at least one (x, y) pair.
- $\forall x \exists y\ P(x,y)$ For every value of x we can find a (possibly different) y so that P(x,y) is true.
- $\exists x \forall y\ P(x,y)$ There is at least one x for which P(x,y) is always true.

Suppose P(x,y) = “x’s favorite class is y.”

Quantification order is not commutative in general !

Negation of Logical Expressions with Quantifiers

- What about $\overline{\forall x\, P(x)}$ and $\overline{\exists x\, P(x)}$?

- Since the quantifiers are the same as taking a bunch of AND's (∀) or OR's (∃) we obtain applying De Morgan's laws:

$$\overline{\forall x\, P(x)} \equiv \overline{\left(P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n)\right)} \equiv$$
$$\overline{P(x_1)} \vee \overline{P(x_2)} \vee \ldots \vee \overline{P(x_n)} \equiv \exists x\ \overline{P(x)};$$

$$\overline{\exists x\, P(x)} \equiv \overline{\left(P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n)\right)} \equiv$$
$$\overline{P(x_1)} \wedge \overline{P(x_2)} \wedge \ldots \wedge \overline{P(x_n)} \equiv \forall x\ \overline{P(x)}$$

Negation of Logical Expressions with Quantifiers

$$\overline{\forall x\, P(x)} \equiv \exists x\, \overline{P(x)} \qquad \overline{\exists x\, P(x)} \equiv \forall x\, \overline{P(x)}$$

- Negation of the expression containing the universal quantifier like "for every x belonging to some certain domain P(x) holds" means "there exist at least one x in the certain domain such that P(x) does not hold"
- Negation of the expression containing the existential quantifier like "there exist such x in the certain domain that P(x) holds" means "for every x from the certain domain P(x) does not hold"

Negation of Logical Expressions with Quantifiers

- General rule: to negate a logical expression containing quantifier (quantifiers), move negation to the expression under quantifier (quantifiers) and flip all quantifiers from $\forall$ to $\exists$ and vice versa.

Negation Example

Compute: $\overline{\forall x \exists y \left(x^2 \leq y \right)}$

- In English, we are trying to find the opposite of "every x admits a y greater or equal to x squared". The opposite is that "some x does not admit y greater or equal to x squared"
- Algebraically, one just flips all quantifiers from ∀ to ∃ and vice versa, and negates the interior propositional function. In our case we get:

$$\overline{\forall x \exists y \left(x^2 \leq y \right)} \equiv \exists x \forall y \overline{\left(x^2 \leq y \right)} \equiv \exists x \forall y \left(x^2 > y \right)$$

Logical Equivalences Involving Predicates and Quantifiers

- Statements involving predicates and quantifiers are logically equivalent if and only if they have the same truth value no matter which predicates are substituted into these statements and which domain is used for the variables in these propositional functions.