CMPSCI 250: Introduction to Computation

Lecture 7: Quantifiers and Languages

David Mix Barrington and Ghazaleh Parvini

20 September 2023

Quantifiers and Languages

- Repeat Three Slides from Lecture 6
- Quantifier Definitions
- Translating Quantifiers
- Types and the Universe of Discourse
- Some Quantifier Rules
- Multiple Quantifiers
- Languages and Language Operations
- Language Concatenation and Kleene Star

Viewing Functions as Relations

- In computing, we usually think of a function from A to B as an entity that takes **input** of some particular type A and produces **output** of some particular type B.
- The formal mathematical definition of "function", however, is different.
- In calculus, they may have tried to impress upon you that a function from $\mathbb{R}$ to $\mathbb{R}$ is a **set of ordered pairs** of real numbers.

Viewing Relations as Functions

- A curve in Cartesian coordinates may ($y = x^2$) or may not ($x = y^2$) represent a function, depending on whether it gives a unique output value for every input value. If it is not a function it is "just a relation".
- This is precisely the language we are using here. The set of points on the graph is a set of ordered pairs in $\mathbb{R} \times \mathbb{R}$, a binary relation.

Functions in Mathematics

- In mathematics, a relation from A to B is called a **function from A to B** if for every element a of A, there is *exactly one* element b such that (a, b) is in the relation.

- Remember that “exactly one” means both “at least one” and “not more than one”. Given an input value a, there must be some value b such that (a, b) is in the relation, but there *may not* be two such values.

The Existential Quantifier

- Suppose that P(x) is a predicate, where x is a variable of type T.
  For example, T might be a set of dogs and P(x) might mean “dog x is a poodle”.
- The **quantified statement** ∃x: P(x) means “there exists a dog x such that x is a poodle”, or “there is at least one poodle in T”. The symbol “∃” is called the **existential quantifier**.

Universal Quantifiers and Binding

- The quantified statement ∀x: P(x) means "for all dogs x, x is a poodle" or "every dog in T is a poodle".
  The symbol ∀ is the **universal quantifier.**

- Each quantifier **binds** a free variable, making it a **bound variable.** Both the statements ∃x: P(x) and ∀x: P(x) are propositions, as they have no free variables -- they are either true or false once T (the type) and P are defined.

Translating Quantifiers

- We translate quantified statements into English very carefully and mechanically.
  After making a first translation we can adapt to something that sounds more natural.

- In translating “∃x: P(x)”, we say “there exists an x” for “∃x”, “such that” for the colon, and then translate P(x). If we want to emphasize the type of x, we might say “there exists an x of type T such that P(x) is true”. In our example, this was “there exists a dog x such that x is a poodle”.

Translating Quantifiers

- In translating "∀x: P(x)", we say "for all x" for "∀x", nothing for the colon (it becomes a comma), and then translate P(x).
  Again we may emphasize the type -- "for all x of type T, P(x) is true".
  In the example, "for all dogs x, x is a poodle".

- If there are multiple quantifiers the rules for translating the colon change a bit. We translate "∃x: ∃y: P(x) ∧ P(y)" as "there exist a dog x and a dog y such that both are poodles".

Types, Universe of Discourse

- The type of the bound variable is an important part of the meaning of a quantified statement.
- Every variable is typed, and “there exist” and “for all” refer to the type, whether or not we state this in our translation.
- Traditionally logicians have referred to the type as the **universe of discourse** for the variable.

Types and Universal Quantifiers

- This is particularly important for universal quantifiers.
- The statements “all deer have antlers” and “all animals have antlers” have different meanings but might both be written ∀x: A(x) -- the difference would be the type of the variable x. In the first the type of x is “deer”, in the second it is “animals”.

Quantifiers and Empty Types

- We can quantify over types that contain *no* elements -- let's take the set U of unicorns as our example.
- Any statement of the form ∃x: P(x) is false if the type of x is U, as it says "there exists a unicorn such that" something. But any statement of the form ∀x: P(x) is true.
- It is true that all unicorns are green, and also true that all unicorns are not green. (For that matter, it is true that all unicorns are both green and not green -- ∀x: G(x) ∧ ¬G(x) in symbols.)

Some Rules for Quantifiers

- Whenever our original predicate has more than one free variable, we need more than one quantifier to bind them and form a proposition.
  Let D be a set of dogs, C be a set of colors, and let H(d, c) mean “dog d has color c”.
- If I say ∃d: ∃c: H(d, c), this means “there exists a dog d and a color c such that d has c”. Note that the first colon translates as “and” rather than as “such that”.

Quantifiers of the Same Kind

- If instead of ∃d: ∃c: H(d, c), we said ∃c: ∃d: H(d, c), this would mean exactly the same thing. Similarly ∀d: ∀c: H(d, c) and ∀c: ∀d: H(d, c) both mean “every dog has every color”.
- We can switch *similar* adjacent quantifiers, but we will soon see that switching *dissimilar* quantifiers changes the meaning.

Quantifier DeMorgan Rules

- We have two "Quantifier DeMorgan" rules to relate quantifiers to negation.
  We can simplify ¬∃x: P(x) as ∀x: ¬P(x), and ¬∀x: P(x) as ∃x: ¬P(x).

- A universal statement is true if and only if there is not a **counterexample** to it.

- This rule explains the convention about empty types: "All unicorns are green" is equivalent to "there does not exist a non-green unicorn" which is clearly true.

Clicker Question #1

- Consider the statement "It is not true that every dog either likes to bark or likes to chew (or both)." Which of these statements is *equivalent* to it?
- (a) Every dog either likes to bark or likes to chew (or both).
- (b) Some dog dislikes either barking or chewing.
- (c) Every dog likes both barking and chewing.
- (d) Some dog dislikes both barking and chewing.

Not the Answer

Clicker Answer #1

$\sim\forall x{:}B(x)\vee C(x)$

- Consider the statement "It is not true that every dog either likes to bark or likes to chew (or both)." Which of these statements is *equivalent* to it?
- (a) Every dog either likes to bark or likes to chew (or both). $\forall x{:}B(x)\vee C(x)$
- (b) Some dog dislikes either barking or chewing. $\exists x{:}\sim B(x)\vee\sim C(x)$
- (c) Every dog likes both barking and chewing. $\forall x{:}B(x)\wedge C(x)$
- (d) Some dog dislikes both barking and chewing. $\exists x{:}\sim B(x)\wedge\sim C(x)$

Multiple Quantifiers

- Let's look more closely at the effect of multiple dissimilar quantifiers.
  Let x and y be of type `natural` and consider $x \leq y$, which has two free variables.
- If we say $\exists x$: $x \leq y$, this statement still has y as a free variable, so its meaning depends on y. It says that there is a natural less than or equal than y, and this statement is true for any y. (For example, x could be y itself).

Multiple Quantifiers

- Similarly $\exists y$: $x \leq y$ has one free variable, x, and is true for any x.
- We can also form $\forall x$: $x \leq y$, which is never true for any y, and finally $\forall y$: $x \leq y$ which is true if $x = 0$ but false for any other x.
- Now we can make propositions from any of these four statements by quantifying the remaining free variable.

Making Propositions

- The statements ∃x: ∃y: x ≤ y and ∀x: ∀y: x ≤ y are true and false respectively, and can have their quantifier order switched.
- More interesting are ∀y: ∃x: x ≤ y (true), ∀x: ∃y: x ≤ y (true), ∃y: ∀x: x ≤ y (false), and ∃x: ∀y: x ≤ y (true, as x could be 0).
- The second and third examples show that switching dissimilar quantifiers can change the meaning.

Clicker Question #2

- Let's now change our data type from natural numbers to the set $\mathbb{Z}$ of *all* integers, including negative integers. Which of the following four quantified statements is *true*?

- (a) $\forall x: \exists z: (x < z) \wedge (x > z)$
- (b) $\forall x: \forall z: (z < x)$
- (c) $\forall x: \forall y: (x \leq y) \wedge (y \leq x)$
- (d) $\forall x: \exists y: (y < x)$

Not the Answer

Clicker Answer #2

- Let's now change our data type from natural numbers to the set $\mathbb{Z}$ of *all* integers, including negative integers. Which of the following four quantified statements is *true*?

- (a) $\forall x: \exists z: (x < z) \wedge (x > z)$ z can't both be >x and <x
- (b) $\forall x: \forall z: (z < x)$ all numbers less than all numbers?
- (c) $\forall x: \forall y: (x \leq y) \wedge (y \leq x)$ all numbers equal?
- (d) $\forall x: \exists y: (y < x)$ but not true for the naturals

Languages, Language Operations

- Recall that for any finite alphabet Σ we have defined the set Σ^* of all **strings** made up of a finite sequence of letters from Σ, and defined a **language** over Σ to be any subset of Σ^*, that is, any set of strings. Here we'll have $\Sigma = \{a, b\}$.
- Because languages are sets, we can use any of our set operators on them.

Set Operators on Languages

- If X is all strings beginning with a, and Y is all strings ending in b, then X ∪ Y is the set of all strings that begin with a *or* end in b, and X ∩ Y is the set of all strings that *both* begin with a *and* end in b.
- Similarly, we can define X Δ Y, X ∖ Y, and the complements of X and Y respectively. For example, the complement of X is the set of all strings that don't begin with a (including the empty string λ).

More Language Operations

- Now that we have quantifiers, we will be able to define two more operations on languages, called **concatenation** and **Kleene star**.
- In the last third of the course, we'll use these two operations and the union operation to define **regular expressions** and thus define the class of **regular languages**.

Language Concatenation

- We'll now define the **concatenation product** (or just **concatenation**) of two languages.
- Remember that the concatenation of two strings is what we get by writing the second string after the first.
- In general, XY is the language $\{w: \exists u: \exists v: (w = uv) \wedge (u \in X) \wedge (v \in Y)\}$. A string w is in XY if it is possible to split it as a string in X followed by a string in Y.

Concatenation Example

- Again let X = {w: w begins with a} and Y = {w: w ends in b}.
- The product XY is the set of all strings that we can make by writing a string in X followed by a string from Y.
- In this example, XY is the same language as X ∩ Y. Any string in XY must both begin with a and end with b, and any string with these two properties can be split into a string in X and a string in Y.

Properties of Concatenation

- Concatenation is *not* commutative, unlike most "multiplication" operations. The language YX is $\{w: \exists u: \exists v: (w = uv) \wedge (u \in Y) \wedge (v \in X)\}$. Strings in YX need not begin with a or end in b -- in fact a string is in YX if and only if it has a letter b that is immediately followed by an a.

- If we let "a" and "b" denote the languages {a} and {b}, with one string each, what is the language $a\Sigma^*b$? Or $\Sigma^*ba\Sigma^*$?

Clicker Question #3

- Which string *is not* in the language denoted by "Σ^*abaΣ^*abaΣ^*"?
  - (a) ababbabab
  - (b) abababab
  - (c) abaabbabba
  - (d) ababbabbaba

Not the Answer

Clicker Answer #3

- Which string *is not* in the language denoted by "Σ*aba Σ*aba Σ*"?
- (a) **aba**bb**aba**b
- (b) **aba**b**aba**b
- (c) abaabbabba
- (d) **aba**bbabb**aba**

Powers of Languages

- In algebra we say "x^k" to denote the product of k copies of x. Similarly in language theory, if X is a language, we abbreviate the concatenation product XX as "X^2", XXX as "X^3", and so forth.
- It turns out that if we treat concatenation as "multiplication" and union as "addition", the distributive law holds, and using algebraic rules, we can get facts like $(X + Y)^2 = X^2 + XY + YX + Y^2$. (We can't say "2XY" because XY and YX are not guaranteed to be equal.)

The Kleene Star Operation

- X^0 is a special case -- "not multiplying" gives us the multiplicative identity, which turns out to be the language {λ}. (Check that {λ}X = X for any language X.)

- It's convenient sometimes to talk about the language $X^0 + X^1 + X^2 + X^3 + \ldots$, the set of all strings that can be made by concatenating together *any number* of strings from X. We call this language X^*, the **Kleene star** of X.

We've used this notation already when we defined Σ^* to be the set of all strings from Σ.