COMPSCI 250: Introduction to Computation

Lecture #6: Predicates and Relations
David Mix Barrington and Ghazaleh Parvini
18 September 2023

Predicates and Relations

- Last Three Slides From Lecture #5
- Statements That Include Variables
- Signatures and Templates
- Predicates Viewed as Boolean Functions
- Sets of Pairs and Tuples
- Storing Relations in a Computer
- Viewing Functions as Relations

An Example: Exercises 1.8.3-4

- Let P be the compound proposition $p \wedge q$ and let C be $p \vee q$. Of course we could verify $(p \wedge q) \rightarrow (p \vee q)$ by truth tables, but let's look at how to approach the problem using our various strategies.
- Neither trivial nor vacuous proof will work. Let's try Hypothetical Syllogism. If we pick p as our intermediate goal, we can get from $p \wedge q$ to p by Left Separation, and from p to $p \vee q$ by Right Joining.

Example: Proof By Cases

- Let's try Proof by Cases, with p as the intermediate proposition. If we assume that p is true, we can prove p ∨ q by Right Joining, and this gives us a trivial proof of the original implication.
- On the other hand, if we assume that p is false, then it is easy to show that p ∧ q is false, giving us a vacuous proof of the original.

Example: Proof by Contradiction

- Using Proof by Contradiction, we assume both $p \wedge q$ and $\neg(p \vee q)$. The second assumption turns to $\neg p \wedge \neg q$ by DeMorgan.
- Once we have "$p \wedge q \wedge \neg p \wedge \neg q$", it's pretty straightforward to get 0. We use associativity and commutativity to get $(p \wedge \neg p) \wedge q \wedge \neg q$. We have $p \wedge \neg p \leftrightarrow 0$ by Excluded Middle, and our 0 rules say that $0 \wedge x \leftrightarrow 0$ for any x.

Statements that Include Variables

- In propositional logic we view the atomic statements as either true or false -- we do not consider semantic relationships between statements.
- There is a similarity between “Cardie is a terrier” and “Biscuit is a terrier” that we may want to capture -- if we are told that “all terriers are dogs” then both of the statements will have consequences.

Predicates

- Rather than give a name to *each* proposition, we might write T(c) and T(b), using the predicate T(x) to mean "x is a terrier". Also define D(x) to mean "x is a dog". Then our new statement would be that $T(x) \rightarrow D(x)$ is *always* true.
- Formally a **predicate** is a statement that *would* become a proposition if the value of some variable is supplied.
- The variables needed to define the meaning of the predicate are called **free variables**.

Signatures and Templates

- Methods in Java have **signatures** that indicate what arguments they take, and in what order. If we define `int foo (int x, String w)`, we know that the method foo takes two arguments, the first an int and the second a String.
- If then a method call `foo (7, "Cardie")` is made, we know that the code of foo will be run, with occurrences of x replaced by 7, and occurrences of w replaced by "Cardie".

Signatures and Templates

- Predicates also have signatures that indicate how many free variables there are, what types they have, and what order they come in.
- A predicate also has a **template**, which is a statement about the free variables. (Often the types of the free variables are made clear in the template.) When we evaluate the predicate for particular objects, we substitute them for the corresponding free variables in the template and thus get a proposition that is true or false. We can think of the predicate as a *function with boolean output*.

Predicates as Boolean Functions

- In mathematics, we define general functions in the same way. In a calculus course, we might say "let f(x) be $3x^2 + 6x + 2$" and then later talk about $f(x + \Delta x)$ by which we mean $3(x + \Delta x)^2 + 6(x + \Delta x) + 2$.
- The signature of this function is "x", a real variable, and the template is the expression $3x^2 + 6x + 2$.

Arity of Predicates

- Just like functions may have different numbers of arguments, predicates may have different numbers of free variables.

- A predicate is:
  unary if it has a single free variable,
  binary if it has two free variables,
  ternary if it has three free variables,
  and in general **k-ary** if it has k free variables.

- The arity of a predicate is its number of free variables.

Return Values of Predicates

- A mathematical or Java function may have any **return type**, but predicates always return booleans. Thus if the template involves numbers, it should also include a boolean operator like $=$ or $\leq$.

- In fact these operators are *themselves* predicates, although we use **infix notation** rather then the usual functional notation.
  For example, we might write $x \leq y$ as LE(x, y).

More Predicates

- When we write x < y, we are giving an expression that will become a boolean once the values of x and y are supplied. Operators like < are **overloaded** for different argument types.
- Set builder notation involves a predicate, e.g., {x: x is a terrier}. The set is the collection of values (from the correct type) that make the predicate true.

Ordered Pairs

- If A and B are any data types, we can define **ordered pairs** of the form (a, b), where a is from A and b is from B. The pair is ordered because (a, b) is different from (b, a).
  (In fact (b, a) is not an ordered pair of that type, unless A = B.)

- Two ordered pairs (a, b) and (c, d) are *equal* if and only if a = c and b = d.

Direct Products

- We can similarly make ordered triples, ordered 4-tuples, or ordered k-tuples. A point in three-dimensional Euclidean space is represented by an ordered triple where each element is a real number, such as $(2, \pi, -4.6)$.
- The set of all ordered pairs with first element from A and second element from B is called **direct product** of A and B, denoted $A \times B$. Similarly we can have direct products of more than two sets -- Euclidean 3-space is the product $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$.

Cardie (R.I.P. 2008-2020) and Duncan (R.I.P. 2009-2021)

Blaze (b. June 2020) and Rhonda (b. Nov 2021)

Practice Clicker Question #1

- Let D be the set {Cardie, Duncan, Rhonda} and let B be the set {Golden, Terrier}. Which of these sets *is not a subset of* the direct product D $\times$ B?

- (a) {(Cardie, Golden), (Cardie, Terrier)}
- (b) {(Cardie, Golden), (Duncan, Terrier)}
- (c) {(Cardie, Golden), (Terrier, Rhonda)}
- (d) {(Cardie, Golden), (Rhonda, Golden), (Duncan,Terrier), (Rhonda, Terrier)}

Not the Answer

Practice Clicker Answer #1

- Let D be the set {Cardie, Duncan, Rhonda} and let B be the set {Golden, Terrier}. Which of these sets *is not a subset of* the direct product D × B?

- (a) {(Cardie, Golden), (Cardie, Terrier)}
- (b) {(Cardie, Golden), (Duncan, Terrier)}
- (c) {(Cardie, Golden), (Terrier, Rhonda)}
  (wrong types in second pair)
- (d) {(Cardie, Golden), (Rhonda, Golden), (Duncan,Terrier), (Rhonda, Terrier)}

Predicates and Tuples

- Let D be a set of dogs and C a set of colors. Let the predicate P(d, c) with signature (D, C), have the template "Dog d has color c".
- Now consider the direct product $D \times C$. An element (d, c) of $D \times C$ is exactly what we need to supply to determine whether P is true or false. (We could think of P as having one free variable, whose type is $D \times C$.)

Relations

- The **relation** corresponding to P is the set {(d, c): P(d, c) is true} -- the set of pairs that make P true. Any subset of D × C is a relation “from D to C”.
- Every predicate has a corresponding relation, and every relation has a corresponding predicate. For example, if $X \subseteq D \times C$, we can define a predicate $P_X(d, c)$ with the template “$(d, c) \in X$”.

Arity of Relations

- Recall: a predicate is unary, binary, ..., k-ary depending on its number of free variables.
- Likewise, a relation is:
  unary if it is just a subset of a single set,
  binary if it is a set of ordered pairs, **ternary** if it is a set of ordered triples,
  and in general **k-ary** if it is a set of ordered k-tuples.
- The arity of a relation and its corresponding predicate is the same.

Practice Clicker Question #2

- Let D be a set of dogs, B be a set of breeds, and let P be the predicate with template “dog x is a Golden but is not the same dog as Cardie”. What set is the relation for this predicate?
- (a) the set of breeds {“Golden”}, unless Cardie herself is a Golden, in which case the set is empty
- (b) the set of all pairs containing the breed “Golden” and some dog other than Cardie
- (c) the set of dogs who are either Goldens or who are not Cardie
- (d) the set of all dogs who are Goldens, except for Cardie

Not the Answer

Practice Clicker Answer #2

- Let D be a set of dogs, B be a set of breeds, and let P be the predicate with template "dog x is a Golden but is not the same dog as Cardie". What set is the relation for this predicate?
- (a) the set of breeds {"Golden"}, unless Cardie herself is a Golden, in which case the set is empty
- (b) the set of all pairs containing the breed "Golden" and some dog other than Cardie
- (c) the set of dogs who are either Goldens or who are not Cardie
- (d) the set of all dogs who are Goldens, except for Cardie

Why not a set of pairs?

- Template “dog x is a Golden but is not the same dog as Cardie”.
- (b) the set of all pairs containing the breed “Golden” and some dog other than Cardie
- (d) the set of all dogs who are Goldens, except for Cardie
- To define what the template *means*, we need to know which dog x is, nothing else. So the relation is a set of dogs, not a set of pairs.

Storing Relations on Computers

- There are three basic ways to store a predicate/relation on a computer.
- We can have an **array of boolean values** where for every possible choice of the predicate's free variables, there is a boolean saying whether that choice is in the relation.
- We can have a **boolean method** that takes the values as arguments and computes a boolean telling whether the k-tuple of arguments is in the relation.

Storing Relations on Computers

- We can have a **list of the tuples** in the relation, so we test the predicate by seeing whether the particular tuple is in the list.
- There are tradeoffs among these three methods.
  For example, the boolean array is generally the fastest but takes the most storage.

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.

Practice Clicker Question #3

- Let D be a set of dogs {Cardie, Duncan, Rhonda} and let B be a set of breeds {Terrier, Golden}. Which of the following *is* an example of a relation from D to B that *is* a function? (We denote elements by their first letters)
- (a) {(c, g), (d, t)}
- (b) {(r, g), (c, g), (d, t), (c, t)}
- (c) {(c, g), (c, t), (d, g)}
- (d) {(c, g), (r, t), (d, t)}

Not the Answer

Practice Clicker Question #3

- Let D be a set of dogs {Cardie, Duncan, Rhonda} and let B be a set of breeds {Terrier, Golden}. Which of the following *is* an example of a relation from D to B that *is* a function?
  (We denote elements by their first letters)
  - (a) {(c, g), (d, t)} (no breed for r)
  - (b) {(r, g), (c, g), (d, t), (c, t)} (two breeds for c)
  - (c) {(c, g), (c, t), (d, g)} (two for c, none for r)
  - (d) {(c, g), (r, t), (d, t)}