CS 2742 (Logic in Computer Science) – Fall 2008 Lecture 20

Antonina Kolokolova

November 10, 2008

7.1 Relations

Recall that a relation on n variables is just a subset of a Cartesian products of n sets which are domains of the variables. In this respect, a binary relation on a set A is a just a subset of $A \times A$.

For example, $R(x, y) \subseteq \mathcal{Z} \times \mathcal{Z}$ such that $R(x, y)$ if and only if $x \leq y$ is a binary relation. In this case, we often write directly $x \leq y$, rather than writing $R(x, y)$. The relation $x = y$ is also a binary relation, which can be defined for many different domains such as integers, reals, strings and so on.

Another common binary relation is congruence $\mod n$ for a natural number n: in that case, $R(x, y)$, written as $x \equiv y \mod n$, if $\exists z \in \mathcal{Z}$ such that $x - y = zn$. This is the same as saying that x and y have the same remainder from division by n. For every n there is a different $\mod n$ relation.

Yet another binary relation is $Parent(x, y)$, which contains all pairs x, y such that x is a parent of y. At this point you may ask what is the difference between a predicate $Parent(x, y)$ and a binary relation $Parent(x, y)$: the predicate is true on the pairs (x, y) that belong to the set which is the binary relation.

There are several major types of binary relations. In this case, it is often more convenient to view a binary relation $R(x, y)$ as "taking" a x "into" y. Binary relations can be:

- Reflexive: $\forall x \in A \ R(x, x)$.
  For example, $x = y, x \leq y, x \equiv y \mod n$ are reflexive, but $Parent9(x, y)$ and $x < y$ are not.
- Symmetric: $\forall x, y \in A \ R(x, y) \rightarrow R(y, x)$.

Antisymmetric: $\forall x, y \in A\ R(x,y) \wedge R(y,x) \rightarrow x = y$.
For example, $x = y, x \equiv y \mod n$ are symmetric, $x \leq y$, $x < y$ and $Parent(x,y)$ are antisymmetric, and the relation $Likes(x,y)$ defined on the domain of people is neither symmetric nor antisymmetric. Note that symmetric and antisymmetric are not complements: an easy way to see that is by noticing that both have a universal quantifier in the definition, whereas a complement of a universal quantifier is existential quantifier. Another way of checking it is to find a relation such as $Likes(x,y)$ which is neither symmetric nor antisymmetric.

- Transitive: $\forall x, y, z \in A\ R(x,y) \wedge R(y,z) \rightarrow R(x,z)$
For example, relation $x = y$ is transitive, because if $x = y$ and $y = z$ then $x = z$ as well. Same can be said about $x \leq y$ and $x < y$: if x is smaller than y, and y is smaller than z, then x is definitely smaller than z.

  Note that $Parent(x,y)$ relation is not transitive: if x is a parent of y, and y is a parent of z, then x is not a parent of z, it is a grandparent. But often we do want to express the fact that one person is related to another by a chain of Parent(x,y) relationships: that is, it is a grandparent, or great-grandparent, or great-great-grandparent and so on. For that, we can define a relation $Ancestor(x,y)$, which would contain all pairs (x,y) related by a chain of $Parent()$ relations. Such a relation is called *a transitive closure.*

  Definition 1. *For a relation $R(x,y)$, its transitive closure contains all pairs x, y such that there is a sequence $z_1, \ldots, z_n$ with $R(x, z_1)$, $R(z_n, y)$ and $\forall 1 \leq i < n\ R(z_i, z_{i+1})$.*

  For example, if John is a grandparent of Jill via Mary who is the daughter of John and the mother of Jill, then x=John, y=Jill and $n = 1$, so there is just one z_1=Mary such that Parent(John,Mary) and Parent(Mary, Jill). Thus, Ancestor(John, Mary) holds.

  A way to describe the Ancestor relation is by the following recursive definition (of the Ancestor predicate): $Ancestor(x,y) = Parent(x,y) \vee \exists z Parent(x,z) \wedge Ancestor(z,y)$.

  However, this definition is not a first-order formula – it mentions the relation $Ancestor(x,y)$ which is being defined. Moreover, as we have seen with the Flight example in the predicate logic, transitive closure cannot be defined by a first-order formula.

A relation that is reflexive, symmetric and transitive is called an *equivalence* relation. Equality and congruence mod 2 are equivalences.So is equivalence of digital circuits computing the same function. Any equivalence relation breaks up the set of all objects on which it is defined into equivalence classes: in such a class all objects are equivalent to each other, but not to any object outside of the class. For example, $x = y \mod 5$ breaks all natural numbers into 5 equivalence classes: 1) numbers that are divisible by 5, 2) numbers that have a remainder 1 from division by 5, 3) ones with remainder 2, 4) with remainder 3, 4) with remainder 4. In the first class, there are numbers 0,5,10,15, 20.., in the third – 2,7,12,17,22,... It is possible to extend this definition to all integers: -1 will be in the same class as 4, -2 in the same class as 3, -5 as 5 and so on. For the names of the equivalence classes we can pick any element of them, but it is convenient to choose $0, 1, 2, 3, 4$, and in general, we represent equivalence classes $\mod n$ as $0, 1, \ldots, n-1$.

There is a special term to denote relations which are reflexive, transitive and antisymmetric: they are called *partial order* relations (e.g., subset relation). A total order is a subclass of partial orders with an additional property that any two elements are related: that is, for any x, y either R(x,y) or R(y,x). E.g.: $\leq$ on numbers.

7.2 Application to cryptography

Let's look more at the congruence modulo n relation.

Enumerate all letters: A=1, B=2, C=3, D=4,E=5,F=6, ..., Z=26. In Caesar's cipher, a letter gets encoded by a letter which is 3 places further in the alphabet. For example, the letter B gets encoded as E, A as D and so on. So the formula seems to be $Code(i) = Letter(i+3)$. But this formula does not tell us how to encode, say, letter Z. Intuitively, we want to "wrap around" and encode Z as C. How do we represent it by a formula? Here is where $\mod n$ relation is useful. In order to represent "wrap around", it is enough to write $Code(i) = Letter(i+3) \mod 26$, where 26 is the number of letters. Now, to decode we just take $Letter(i) = Code(i-3) \mod 26$ (remember that our definition of $\mod$ can handle negative numbers).

The problem with Caesar's cipher is that seeing enough messages one can figure out the code, and knowing how to encode can easily decode. In the next lecture we will see some more secure cryptosystems.