Equivalence Relations

Margaret M. Fleck

20 November 2009

This lecture covers material on equivalence classes from section 8.5 of Rosen, a topic we'll continue through Monday.

1 Announcements

Reminder: quiz Wednesday after break.

2 Sketch of ideas

Equivalence relations are relations that are reflexive, symmetric, and transitive (RST). They are like equality except that they treat certain groups of elements as equivalent to one another.

For example, consider congruence mod 12. The congruence relation treats all numbers with remainder zero (0, 12, -12, 24, ...) as if they were equal. Similarly, the numbers with remainder 3 (3, -9, 15, -21, ...) are treated as equal. We have divided up the integers into 12 subsets, which don't overlap with one another. This is called a partition of the set of integers.

Let's name the subsets by their remainders: [0], [1], ... [11]. So, these are like the 12 hours on the dial of a clock. After [11], you go back to [0] again. Or maybe you go to [12] and then to [1] after that. One hour has two names: 0 or 12. Others have multiple names if you move back and forth between

12 and 24 hour clocks, e.g. 1800h is the same as 6pm. I.e. [18] and [6] are names for the same hour.

We can now treat each subset as if all its members were merged into a single object and define operations on these merged objects, using modular arithmetic. For example, if we add 2 hours to 11 o'clock, we get 1 o'clock. In our new notation: $[2] + [11] = [1]$. In general:

$$[x] + [y] = [(x + y) \bmod 12]$$

We can now give a friendly name to this new set of objects. In this case, it's the integers mod 12, written $\mathbb{Z}_{12}$.

This trick is widely used in mathematics to create new sets of objects such as modular integers, the real numbers, the rationals. It's even used to do a mathematical version of cut-and-paste to create strange sorts of geometrical objects such as Möbius strips.

This construction only works for equivalence relations because, as we will see, the three properties of an equivalence relation are needed to guarantee that we can neatly partition up the set. Also, we need to show which operations on the input set (e.g. addition) can be made to work right on the new subset-objects.

3 Equivalence classes

Let's define some terminology. Suppose that R is an equivalence relation on a set A. For each element $x \in A$, we define the *equivalence class of* x to be the set of all elements related to x. That is

$$[x]_R = \{y \in A \mid xRy\}$$

(The R subscript on the square brackets is only written when we want to be especially clear about which relation is being used.)

So, for example, if the relation is congruence mod 12, here are some equivalence classes

$$[2] = \{\ldots, -22, -10, 2, 14, 26, \ldots\}$$
$$[6] = \{\ldots, -18, -6, 6, 18, \ldots\}$$
$$[18] = \{\ldots, -18, -6, 6, 18, \ldots\}$$

Notice that [6] and [18] are equal. That is, they are two names for the same subset of the integers. Other names for this same subset include [40], and [−18]. To refer to an equivalence class, we pick any element of the class as its *representative*.

Every equivalence relation has a corresponding set of equivalence classes. For example, suppose we define a relation T on pairs of integers by $(x, y)T(p, q)$ if and only if $x^2 + y^2 = p^2 + q^q$. Then $[(1, 0)]$ is the circle with radius 1 centered at the origin. And, in general, each equivalence class is a different circle centered at the origin, except for $[(0, 0)]$ which is a single point.

Or, suppose we consider all functions from the reals to the reals. We can then define $f \sim g$ if and only if $f = \Theta(g)$. That is, we group together all functions with the same big-O growth rate. Then $[x^2]$ contains all functions that grow quadratically, e.g.

$$[x^2] = \{x^2, 3x^2 + 17, 2x^2 + \log x, x^2 + x, \ldots\}$$

We could also make up an equivalence relation so that it follows some type of real-world relationship. For example, suppose we took all the students in this classes and divided you up into teams. Each team would then be an equivalence class. The corresponding relation must then relate two students exactly when they belong to the same team. If Fred and Sally are both members of the same team, we could refer to it as "Fred's team" or also as "Sally's team." People in the class would know that these two names referred to the same team.

4 Defining datatypes with equivalence relations

An important use of equivalence classes is in defining new datatypes. Suppose that someone has given us the integers and we want to define the rational numbers. Let's define F to be the set of fractions with non-zero denominator. A fraction is just a pair of integers, written funny. Now define an equivalence relation $\sim$ on F by

$\frac{x}{y} \sim \frac{p}{q}$ if and only if $xq = yp$.

This is the equivalence relation used to define rational numbers. A fraction $\frac{x}{y}$ represents the same rational number as $\frac{p}{q}$ if and only if $xq = yp$. Then $[\frac{3}{4}]$ and $[\frac{12}{16}]$ are two names for the same rational number.

We can now give a friendly name to our new set: the rational numbers, written $\mathbb{Q}$. And we can define how operations such as addition and multiplication are supposed to work on these new numbers (see next lecture). This mathematical description of the rationals could then be used to implement a new datatype "rational" for your favorite programming language.

For this class, we'll try to be careful about writing the square brackets around equivalence classes. However, once all the basic properties of a datatype have been established, it's not uncommon to drop the brackets. For example, people typically write fractions without the square brackets around it. But the idea hasn't changed: when we say "the rational number $\frac{x}{y}$", we mean $\frac{x}{y}$ to be a representative of the whole set of equivalent fractions.

Similarly, many computer languages have a built-in datatype for strings. These are usually case-sensitive, so that "Uiuc", "uiuc", and "UIUC" would be considered three different objects. For many language-processing applications, we'd like to consider them the same. So, mathematically, we could define an equivalence relation that relates a pair of strings if and only if they are the same except for case differences. Then these three things would all be in the same equivalence class $[uiuc]$.

Similarly, suppose that we're using a computer language which already has a datatype for lists of integers and we want to define a new "set" datatype for finite sets of integers. We can do this by treating certain lists as equivalent.

Specifically, let L contain all finite lists of integers. Then we create an equivalence relation $\sim$ in which two lists $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_n$ are related if two conditions hold

a) For every i, there is a k such that $a_i = b_k$.

b) For every k, there is an i such that $b_k = a_i$.

We would then create a new type name (e.g. Set) for the equivalence classes of lists. Each new Set object would then contain a list, which is some representative of its equivalence class. We've got a few choices here. If we store the list given to us by the user, we need to make sure that the function set-equal returns "true" if we give it two sets whose lists are different but contain the same elements. Alternatively, we might force each Set object to always contain the representative of its class which is in sorted order. This complicates construction of new Set objects but makes set-equal easy to implement.

5 Partitions

When we used a relation R to divide up a set A into equivalence classes, we depended implicitly on the set of classes being a *partition* of the input set of values. That is, the equivalence classes cover the the whole set A, they don't overlap, every equivalence class contains at least one representative element. Being a partition is very convenient because means that we don't have to consider annoying possibilities such as two equivalence classes having a partial overlap.

Formally, a partition of a set A is a collection of non-empty subsets of A which cover all of A and don't overlap. Specifically, if the subsets are $A_1, A_2, \ldots A_n$, then they must satisfy three conditions:

a) $A_1 \cup A_2 \cup \ldots \cup A_n = A$

b) $A_i \neq \emptyset$ for all i

c) $A_i \cap A_j = \emptyset$ for all $i \neq j$.

A partition can contain an infinite set of subsets. To cover this possibility, we need to use a more general notation. Let P be our partition. then the three conditions are:

a) $\bigcup_{X \in P} X = A$

b) $X \neq \emptyset$ for all $X \in P$

c) $X \cap Y = \emptyset$ for all $X, Y \in P$, $X \neq Y$

We've seen how an equivalence relation generates a partition. You can do this construction the opposite way as well, starting with a partition P and using it to construct an equivalence relation $\sim$. Specifically, we define $x \sim y$ if and only if x and y are in the same element of P.

6 Preview

We need to show formally that the equivalence classes of an equivalence relation actually form a partition of the base set. If we started with any random relation, not an equivalence relation, they might not. And then bad things would happen when we tried to do things like defining operations on the equivalence classes. For example, how would we define addition if one of the input equivalence classes had nothing in it?

Next lecture, we'll see that the three defining properties of an equivalence relation are sufficient to ensure that the equivalence classes do form a partition.