Lab: Implementing Relations
CSC/MAT 208: Discrete Structures
Department of Computer Science
Department of Mathematics and Statistics
Grinnell College

Pairs

- Kathryn Yetter and Corey Simmonds
- Jong Hoon Bae and Anita DeWitt
- Matt Jasaitis and Thu Nguyen
- Elizabeth Zak and Dennis Chan
- Kevin Connors and Jacob Ekstrand
- Tyler Williams and Linda Oyolu
- Michael Owusu and David Chang
- Amanda Hinchman-Dominguez and Devin Dooley
- Bazil Mupisiri and Marcel Champagne
- Muhammad Hamza and Madeleine Goldman
- Lindsey Byrne and Julia Fay
- Adam Hudson and Danielle Williams
- Samee Zahid and Ella Nicolson

Adam Wesely will take the place of anyone who is absent

Setup

For this lab, you’ll need the (discrete classes) library (if you don’t already have it) and a “starter” version of the (discrete relations) library. These are available at

/home/stone/discrete/code/discrete/classes.sls
/home/stone/discrete/code/discrete/relations.sls

Copy them into the discrete subdirectory of your DrRacket collections directory. Then start DrRacket and open your copy of the (discrete relations) library.
Representing Relations

There are several ways to represent relations in Scheme, none of which is clearly the best overall. The efficiency of common operations depends on which of the underlying representations is used, and different representations are optimal for different operations. We’ll explore some of the options in future labs.

In this library, a relation is represented as a record with a single field, called pairs. This field contains a class of pairs. A pair that has $x$ as its car and $y$ as its cdr is a member of this class just in case $x$ stands in the represented relation to $y$. So, for instance, in the relation of less to greater among the integers 0, 1, 2, and 3, the pairs field would contain the class of the six pairs $(0 . 1), (0 . 2), (0 . 3), (1 . 2), (1 . 3)$, and $(2 . 3)$.

Within the library, we can form a relation record from a given class of pairs by applying the constructor make-relation-rec to it. Similarly, we can use the selector relation-rec-pairs to recover the class of pairs from a given relation record. The library exports these operations under different names (pair-class->relation and relation->pair-class) so that applications will still be able to use them, but their underlying implementation will not be visible.

Exercise 0. In the Interaction window, construct the relation record for the relation of less to greater among the integers from 0 to 3. You can use the display-relation procedure to show the relation once you have constructed it. Give the relation the name less-than (by embedding the constructor call in a definition).

Relational Analogues of Class Operations

The first section of today’s reading dealt with operations and properties of relations that were analogous to operations on classes: union, intersection, difference, equality, and so on. Read through the definitions of these operations in the new library, and note that, in most cases, the whole procedure consists of unpacking the class of pairs from each relation, performing the analogous class operation on those classes, and then (if necessary) converting the result back into the relation data type.

Exercise 1. I left the relation-sunder procedure for you to implement. It’s slightly more subtle than some of the other relational analogues of class operations, because it takes a binary (two-argument) predicate,
whereas `class-sunder` takes a unary (one-argument) predicate. Thus you can’t just use the same predicate in the underlying call to `class-sunder`.

As a test case, the call

\[
\text{(display-relation (relation-sunder less-than}
\text{ (lambda (left right)
\text{ (= (+ left right) 3))))})
\]

should write out

\{<0, 3>, <1, 2>\}

(possibly listing the pairs in the opposite order). Explain why this is the correct result.

Properties of Relations

The second section of today’s reading introduced a number of properties of relations, such as reflexivity, symmetry, connexity, and transitivity. Most of these are implemented more or less straight from the definitions, by checking that all of the pairs in the underlying class of pairs satisfy some condition. (Reflexivity is slightly exceptional, since the predicate that checks it needs to know the universe of discourse — that is, the class of all values to be considered — as well as the relation to be tested.)

**Exercise 2.** In the Interactions window, find out whether the `less-than` relation that you defined in exercise 0 is reflexive. We can’t supply an infinite class as the universe of discourse in this case; what would be an appropriate choice for the second argument?

**Exercise 3.** I’ve left the tests for symmetry, asymmetry, and antisymmetry for you to implement. They are variations on a theme, and I thought that seeing exactly how they differ would help to fix their meanings in your minds. Define and test these three predicates.

**Exercise 4.** Find out which of the remaining defined properties of relations `less-than` has: irreflexivity, connexity (in what universe of discourse?), transitivity, intransitivity. Is it an equivalence relation? Is it a partial ordering? Is it a total ordering?
Operations on Relations

The third section of the reading introduced a variety of operations that transform relations in various ways: converse, image, relational product, and so on. Read through the definitions of these procedures in the (discrete relations) library.

Exercise 5. With pencil and paper, compute the converse, left domain, and right domain of the less-than relation you defined in exercise 0. Then use appropriate commands in the Interactions window to check your results.

Exercise 6. Define the image and preimage operations and test your definitions.