Properties Related to Finite Functions

As explained in the reading on functions, we count a relation as a function if it associates each value (as argument) in the universe of discourse with one and only one value (as result).

**Exercise 0:** Extend the (discrete relations) library with a predicate `functional?` that takes as its arguments a relation and a class (the universe of discourse within which the relation is defined) and determines whether the given relation is a function in that universe.

There is also a related but slightly broader concept, under which a relation is a *function on its left domain* if it associates each value in the relation’s left domain with one and only one value from the relation’s right domain.
Exercise 1: Extend the (discrete relations) library with a predicate functional-on-domain? that takes just one argument, a relation, and determines whether that relation is a function on its left domain.

Recall that a function is a bijection if, and only if, its converse is also a function.

Exercise 2: Extend the (discrete relations) library with a predicate bijection? that determines whether a given function is a bijection.

Operations on Finite Functions

It would be possible to define new procedures for the common operations of inverting a bijection (that is, reversing the roles of arguments and results), composing functions, and taking the n-th iterate of a function. However, these turn out to be special cases of the general operations on relations that are already defined in the (discrete relations) library.

Exercise 3: (a) What relational operation corresponds to inverting a bijection? (b) ... to composing two functions? (c) ... to taking the n-th iterate of a function?

Other operations on functions are loosely inspired by Scheme primitives. For instance, one could define a function-apply procedure that takes a function and a value in its left domain and returns the corresponding result. (It searches through the pairs stored in the relation func for the one that has arg as its car, then returns the corresponding cdr.)

(define function-apply
  (lambda (func arg)
    (let loop ((pair-list (class->list (relation->pair-class func)))))
      (cond ((null? pair-list) (error 'function-apply "Function domain error"))
            ((equal? (caar pair-list) arg) (cdar pair-list))
            (else (loop (cdr pair-list)))))))

Exercise 4: Add function-apply to the (discrete relations) library, then extend that library with a finite-function analogue of Scheme’s map procedure: Given a list of members of the left domain of a function and the function itself, your function-map procedure should return a list of the results of applying the function to each element of the list in turn.
The Cyclic Structure of Iterated Finite Functions

If \( f \) is a finite function in a finite universe of discourse, then, in the sequence of results obtained by applying the successive iterates \( f^{(0)}, f^{(1)} \), \( f^{(2)} \), and so on, to the same argument \( x \), we will eventually encounter repetitions. For instance, suppose that the universe of discourse is

\[
\{ \text{yellow, green, black, red, blue} \},
\]

and \( f \) is defined by the equations

\[
\begin{align*}
 f(\text{yellow}) &= \text{blue} \\
 f(\text{green}) &= \text{blue} \\
 f(\text{black}) &= \text{red} \\
 f(\text{red}) &= \text{yellow} \\
 f(\text{blue}) &= \text{green}
\end{align*}
\]

A pattern is quickly discernible when we apply successive iterates of \( f \) to green:

\[
\begin{align*}
 f^{(0)}(\text{green}) &= \text{green} \\
 f^{(1)}(\text{green}) &= \text{blue} \\
 f^{(2)}(\text{green}) &= \text{green} \\
 f^{(3)}(\text{green}) &= \text{blue} \\
 f^{(4)}(\text{green}) &= \text{green}
\end{align*}
\]

The repetitions start with \( f^{(2)} \), and it’s apparent that we’re going to cycle back and forth between green and blue as we apply higher-order iterates.

The story’s a little different if we start with, say, black:

\[
\begin{align*}
 f^{(0)}(\text{black}) &= \text{black} \\
 f^{(1)}(\text{black}) &= \text{red} \\
 f^{(2)}(\text{black}) &= \text{yellow} \\
 f^{(3)}(\text{black}) &= \text{blue} \\
 f^{(4)}(\text{black}) &= \text{green} \\
 f^{(5)}(\text{black}) &= \text{blue} \\
 f^{(6)}(\text{black}) &= \text{green} \\
 f^{(7)}(\text{black}) &= \text{blue}
\end{align*}
\]
This time, the repetitions don’t start until we get to $f^{(5)}$, but we’re eventually locked into the same cycle, going back and forth between green and blue.

Depending on the original function, we may or may not always reach the same cycle, and the cycle we do reach doesn’t always contain just two members of the universe of discourse, but in a finite universe of discourse any function, iteratively applied to any value, does eventually lock into some cycle or other, either immediately (as in the example with green above) or after some initial non-repeated steps (as in the example with black). In a sense, the formal structure of the sequence of iterates is determined by two numbers: the length of the cycle that we eventually enter, and the length of the “tail” of non-repeated values before we get into the cycle. Thus the structure of the sequence of iterates of $f$ applied to green is determined by the numbers 2 and 0—a cycle containing two steps, reached after zero unrepeated values; and the structure of the sequence of iterates of $f$ applied to black is determined by the numbers 2 and 3—a cycle of two steps, entered after three unrepeated values.

**Exercise 5:** Define and test a Scheme procedure that takes a finite function `func` and an argument `arg` to that function and returns a pair containing the two numbers that determine the structure of the sequence of iterates of `func` applied to `arg`.

---

I am indebted to Ethan Ratliff-Crain 2015 and Justus Goldstein-Shirley 2016 for pointing out errors in the definition of `function-apply` in an earlier version of this document.

Copyright © 2013, 2014, 2016 John David Stone

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en_US

or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.