Lecture 06: Shadowing and Recursion
10:00 AM, Sep 17, 2018

Contents

1 Review of What We Know So Far .......................... 1

2 Environmental Studies..................................... 1
   2.1 Shadowing ............................................. 1

3 Recursion ...................................................... 2
   3.1 Overhanging Bricks ................................... 2

1 Review of What We Know So Far

We already know most of CS17 Scheme! We have learned:

- Rules of Evaluation (number rule, Boolean rule, symbol rule, list-eval rule, lambda rule, quote rule, cond rule, and rule, or rule)
- Notion of binding symbols to data objects

There are still a couple of things we haven’t yet discussed about CS17 Scheme:

- some intricacies of lambda expressions
- a few built-in procedures

2 Environmental Studies

2.1 Shadowing

Imagine you write a procedure, quadratic-formula, which takes in three arguments, a, b, and c, and returns the result of applying the quadratic formula to the three inputs. Your friend wants to use this procedure in their program.

Your friend has defined the variable b: (define b (quote moo))

What happens when they try to apply your procedure?

Let’s start by considering what we want to happen. First, we don’t want the procedure to fail. That is, in the procedure body we want b to refer to the actual argument, 10. We also want to ensure that after the procedure application is over, we still want the variable b to be bound to the symbol moo. How do we accomplish these two fundamental goals?
The solution is called *shadowing*. When the `quadratic-formula` procedure is applied: `quadratic-formula 3 10 3` the first step is to bind `a` to `3`, `b` to `10`, and `c` to `3`. These new bindings made in procedure application hide, or “shadow”, any existing top-level bindings from the evalpig. The environment contains the top-level bindings as well as the bindings created in applying this procedure, but the evalpig gives priority to the latter bindings. Further, once evaluation has occurred, the evalpig no longer sees the bindings created in the procedure application.

Now say we want to apply our procedure to a new set of numbers: `quadratic-formula 1 5 1)` Will the bindings made in the previous application affect this instance of applying our procedure?

Fortunately, the answer is no! Each time a procedure is applied, a new set of bindings are created for the formal arguments, and these bindings form the context for that particular application of the procedure. Other sets of bindings from previous invocations are not “erased”, but they are simply not looked at by the evalpig.

Now we know that each application of a procedure gets its own formal arguments. This allows us to use one application to help another by having the body of our procedure include an application of the same procedure...

### 3 Recursion

#### 3.1 Overhanging Bricks

The overhang (measured in units of half-brick length) with `n` bricks is

\[
\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + ... + \frac{1}{3} + \frac{1}{2} + \frac{1}{1} \tag{1}
\]

This is called the \(n^{th}\) harmonic number, denoted \(H(n)\).

Suppose we want to know what the fifth harmonic number is. We can find this number using the equation

\[
H(n) = \frac{1}{n} + H(n-1) \tag{2}
\]

From the equation we see that in order to arrive at the fifth harmonic number, we simply add \(\frac{1}{5}\) to the fourth harmonic number. Easy, so long as we know the fourth harmonic number! Well how do we find the fourth harmonic number? We just add \(\frac{1}{4}\) to the third harmonic number! Continuing this pattern, we see that we must begin with the zero-th harmonic number, which is just zero. This allows us to calculate the first harmonic number:

\[
H(1) = \frac{1}{1} + H(0) = 1 + 0 = 0 \tag{3}
\]
Now we can continue this pattern to arrive at our desired answer:

\[
\begin{align*}
H(2) &= \frac{1}{2} + H(1) = \frac{1}{2} + 1 = \frac{1}{2} \\
H(3) &= \frac{1}{3} + H(2) = \frac{1}{3} + \frac{1}{2} = \frac{5}{6} \\
H(4) &= \frac{1}{4} + H(3) = \frac{1}{4} + \frac{5}{6} = \frac{17}{12} \\
H(5) &= \frac{1}{5} + H(4) = \frac{1}{5} + \frac{17}{12} = \frac{217}{60}
\end{align*}
\]

This can be implemented in Racket as follows:

```
(define H)
(lambda (n)
  (cond
   ((zero? n) 0)
   (#true (+ (/ 1 n) (H (- n 1))))))
```

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback)