6.001 SICP Spring 2004 2/17/2004

Tutorial 2

Topics

Substitution model, recursion, orders of growth

Upcoming Deadlines

- Problem set 2 due Tuesday, 2/17 at midnight
- Lecture 5 due Wednesday, 2/18 at 9am
- Project 2 due Friday, 2/27 at 6pm

Substitution Model

The substitution model is a means for describing how an expression, in particular a combination, is evaluated. It should be emphasized that the steps of this model is not exactly what happens in the computer. However, it suffices for now in order to understand various issues of computation.
Let us begin with an example. Suppose we have the following expression:

```
((lambda (x) (* 2 x)) 3)
```

How do we evaluate this? First, we begin by identifying what type of expression this is. Of course, it is a combination. Next, we evaluate all of the subexpressions, in any order. The first subexpression is the special form lambda with value "procedure" (or, as introduced last week, a "double-bubble"). The second subexpression is self-evaluating with value 3. Next, we apply the procedure to the values of its arguments. We can do this explicitly by first writing down the body of the procedure:

```
(* 2   x   )
```

Next, we substitute the value of the arguments for the parameters of the procedure. In this case, we substitute 3 for `x`:

```
(* 2   3   )
```

Finally, we continue with the evaluation of the resulting expression.

Rules of Evaluation

To summarize, the rules of evaluation are as follows:

- If **self − evaluating**, return the value.
- If a **name**, return the value associated with name in environment.
- If a **special form**, do something special (the exception to the rules).
- If a **combination**, then
  - *Evaluate* all of the subexpressions of the combination in any order.
  - *Apply* the operator to the values of the operands (arguments) and return result.
    - If the procedure is a **primitive procedure**, then just do it.
    - If the procedure is a **compound procedure**, then
      - Copy the body of the procedure.
      - Replace each formal parameter with the corresponding actual argument value.
      - Evaluate the resulting expression.

Recursion and Orders of Growth

Problem 1 - The Towers of Hanoi

The Towers of Hanoi problem asks how to move a stack of n disks from the one pole to another pole. There are three poles in the game and the disks start off on the first pole stacked on top of each other. The disks are of different sizes stacked initially in decreasing order with the smallest disk on top. You may move only one disk at a time from one pole to another. However, you may not stack a larger disk on top of a smaller disk. We wish to write a program to print the sequence of steps to solve this problem. Suppose we have the following piece of code defined:

```
(define (my-display k from to)
  (display k)
  (display " ")
  (display from)
  (display " ")
  (display to)
  (newline))
```

First, what does this do? Now, we wish to use this to solve our problem:

```
(define (hanoi n)
  code-goes-here)
```