MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007

Recitation 3, Wed, February 14

Substitution, Recursion Solutions Dr. Kimberle Koile

1. **Substitution**

Consider the example below. Notice that x is used in multiple places. When do we substitute for x and when don't we?

```
(define x-y*y
  (lambda (x y)
    (- x ((lambda (x) (* x x)) y))))
```

Use the substitution model to evaluate the following expression, and write each substitution step.

```
(x-y*y  11  3)
([proc (- x ((λ (x) (* x x)) y))]  11  3)
(-  11  ((λ (x) (* x x)) 3))
(-  11  (* 3 3))
(-  11  9)
```

Value: 2

2. **Recursion**

2.1. a. Implement addition as a recursive algorithm that employs repeated successor (in Scheme, this is the inc function). Hint: check for base case, then recursive case.

```
(define (add x y)
 (if (= x 0)
    y
    (add (dec x) (inc y))))
```

b. Write four substitution steps for (add 3 2)

```
(add 3 2)
(if (= 3 0) 2  (add (dec 3) (inc 2)))
(if #f 2 (add 2 3))
(if (= 2 0)  3 (add (dec 2) (inc 3)))
(if #f 3 (add 1 4))
(if (= 1 0) 4 (add (dec 1) (inc 4))
(if #f 4 (add 0 5))
(if (= 0 0) 5 (add (dec 0) (inc 5))
 5
```

There are a variety of ways to write substitution steps, depending on how much detail is given. In the above example, I've omitted the evaluation of `add,` `dec,` and `inc` to [proc:add], [proc:dec], and [proc:inc], respectively. The goal is just to make sure that you understand how the substitution model works. We'll contrast this model with a different model, the environment model, soon.

Note: The following version is a recursive algorithm; the call to `inc` is deferred. (There's no reason to write the procedure in this way; it's shown here as an example.)

```
(define (add x y)
  (if (= x 0)
      y
      (inc (add (- x 1) y))))

  (add 3 2)
  (if (= 1 0) 2 (inc (add (- 3 1) 2)))
  (if #f 2 (inc (inc (add (- 2 1) 2))))
  (inc (inc (inc (add (- 1 1) 2))))
  (inc (inc (inc (add 0 2))))
  (inc (if (= 0 0) 2 (inc (inc (inc (add (- 0 1) 2))))))
  (inc (if #t 2 (inc (inc (inc (add (- 0 1) 2))))))
  (inc (inc (inc 2)))
  (inc (inc 3))
  (inc 4)
  5
```

2.2. Implement subtraction as a recursive algorithm that employs the `dec` function, which decreases its argument by 1.

```
(define (sub x y)
  (if (= y 0)
      x
      (sub (dec x) (dec y))))
```

2.3 Implement exponentiation through repeated multiplication.

a. recursive algorithm

```
(define (expt x n)
        (if (= n 0)
            1
            (* x (expt x (- n 1)))))
```

```
Example: (expt 3 4)
         (* 3 (expt 3 3))
         (* 3 (* 3 (expt 3 2))
         (* 3 (* 3 (* 3 (expt 3 1)))
         (* 3 (* 3 (* 3 (* 3 (expt 3 0))))
         (* 3 (* 3 (* 3 (* 3 1)))
         (* 3 (* 3 (* 3 3)))
         (* 3 (* 3 9))
         (* 3 27)
         81
```

b. iterative algorithm (Hint: Define a helper function.)

```
(define (expt x n)
      (helper x n 1))

(define (expt-helper x counter result)
        (if (= counter 0)
            result
            (expt-helper x (- counter 1) (* result x))))
```

```
Example: (expt 3 4)                (Note: substitution of helper body omitted for brevity)
         (expt-helper 3 4 1)
         (expt-helper 3 3 3)
         (expt-helper 3 2 9)
         (expt-helper 3 1 27)
         (expt-helper 3 0 81)
```

d. What value is returned for (count2 4)? 4