6.001 SICP – September 17

6001-Processes, Substitution, Recursion, and Iteration

Trevor Darrell  trevor@csail.mit.edu
32-D512

6.001 web page: http://sicp.csail.mit.edu/
        section web page: http://www.csail.mit.edu/~trevor/6001/

• Review
  – Syntax
  – Evaluation
  – Procedures
  – Special Forms
• Scheme processes
• Substitution model
• Recursion
• Iteration

Review-Syntax

Things that make up scheme programs:

• primitive types  23. "hello", #t
• names        +, pi
• combinations (+ 2 3)
   (* pi 4)
• special forms (define pi 3.14)
   (lambda ... )
   (if ... )

Note that special forms are not combinations, even
though they are syntactically similar.
A combination requires that the first subexpression be a
procedure.

Review-Evaluation Rules

• A numeral, string or boolean evaluates to
  itself (number, string, #t, #f).
• A name evaluates to
  the value bound to that name in the environment
• A combination is evaluated as follows:
  1. Evaluate the subexpressions in any order
  2. Apply the value of the operator subexpression to the
     values of the remaining subexpressions.
• define and lambda special forms are special:
  1. A define associates the value of the second argument
     with the name given in the first argument
  2. A lambda expression evaluates to a procedure object
     that stores both the parameter list and procedure body

Review-Special forms

(if <pred> <consequent> <alternative>)
(and <expr1> <expr2> ... <exprN>)
(or <expr1> <expr2> ... <exprN>)
(cond (pred1) <expr1> ... <expr1z>)
   (pred2) <expr2a> ... <expr2z>
   . . .
   (predN) <exprNa> ... <exprNz>
   (else <exprEa> ... <exprEz>))
(begin <expr1> <expr2> <expr3>)

Drill : AND / OR from COND / IF

(and a b c) ===
  (cond [(a (cond b (cond c #t) (else #f)))
        (else #f)])
  (if a
   (if b
    (if c #t
     #f)
    #f))

(or a b c) ===
  (cond (a #t)
        (else (cond (b #t)
                    (else (cond (c #t)
                                  (else #f))))))
  (if a #t
   (if b #t
    (if c #t #f)))

Drill : AND / OR from COND / IF
Review-Procedures

Procedures capture a common form...

(* 3 3), (* 4 4), (* n6 96), ...

(define square (lambda (y) (* y y))
  name of proc. name of part of pattern that varies
or, (define (square y) (* y y))

To apply a procedure given a procedure and argument values:
If it is a primitive procedure, "just do it".
If a compound procedure...

Substitution example

(define (square i) (* i i))
(define (sum-of-squares i j)
  (+ (square i) (square j)))

(sum-of-squares 3 4) ...
(+ (sum-of-squares 3 4) ...
(+ (* 3 3) (* 4 4))
(+ (* 3 3) 16)
(+ 9 16)
25

Substitution model

To apply a compound procedure to its arguments:
evaluate the body with each formal parameter
replaced by the corresponding argument

Some terminology:
• in the definition (define (square i) (* i i) )
i is the formal parameter
(* i i) is the body
• in the application (square 3)
3 is the argument

Drill -- substitution

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

(x*y 11 3) ...
(- 11 ((lambda (x) (* x x)) 3))
(- 11 (* 3 3))
(- 11 9)
2

Recursion Overview

Recursion: Solve a problem by solving a smaller instance
and converting into desired solution

e.g., n! = n(n-1)(n-2)... = n * (n-1)!
when does it end (base case)? 1! = 1

Recursive procedure: a nested call, a procedure that literally
calls itself.
e.g., (define (rec a) ... (rec (+ a 1)) ...)

Recursive process: a process (internal sequence of execution) that calls itself with pending operations

Key question of this lecture: how can an iterative processes
be described by a recursive procedure?

General form of recursive algorithms

• test, base case, recursive case

(define fact
  (lambda (n)
    (if (= n 1) ; test for base case
      1 ; base case
      (* n (fact (- n 1)) ; recursive case )))

• base case: smallest (non-decomposable) problem
• recursive case: larger (decomposable) problem
Fun
Given that inc adds one to its argument and dec subtracts one from its argument

\[
\begin{align*}
\text{inc } x & \rightarrow x + 1 \\
\text{dec } x & \rightarrow x - 1
\end{align*}
\]

what does fun do?

\[
\text{(define fun (lambda (a b) (if (= a 0) b (inc (fun (dec a) b)))))}
\]

Is it recursive?

\[
\text{== "is it a recursive process?"}
\]

Fun
\[
\text{(define fun (lambda (a b)}
\begin{align*}
& \text{(if (= a 0) b} \\
& \hspace{0.5cm} (inc (fun (dec a) b)))))
\end{align*}
\]

Sum-ints recursive
Problem: Recursively sum ints from a to b...
- base case?
- recursive case?
- test?

\[
\text{(define sum-recusive (lambda (first last)}
\begin{align*}
& \text{(if (> first last)}} \\
& \hspace{0.5cm} 0 \\
& \hspace{1cm} (+ \text{ first} \\
& \hspace{1.5cm} (\text{sum-recusive (+ first 1) last)))))
\end{align*}
\]

Iteration
What are the elements of an iteration?
- state
- start
- action
- update
- stop
- return

Problem: Sum ints from a to b...

\[
\text{State is } s \quad ;\text{State}
\]

Starting with \( s \leftarrow 0 \quad ;\text{Start}
\]

\[
\text{a \leftarrow a + 1} \quad ;\text{Update}
\]

\[
\text{repeat} \\
\text{stop when } a > b \quad ;\text{Stop}
\]

The answer is the value of \( s \quad ;\text{Return}
\]

Iteration
What are the elements of an iteration?
- state
- start
- action
- update
- stop
- return

How do we do this in scheme (i.e., in a recursive procedure)?

...use a "helper function":

\[
\text{(define (helper [STATE] )}
\begin{align*}
& \text{(cond ( [DONE] [RETURN] )} \\
& \hspace{0.5cm} \text{(else [ACTION]}
\end{align*}
\]

\[
\text{(helper [UPDATE] )})
\]

\[
\text{(define (main) (helper [START] )})
\]

Iteration
What are the elements of an iteration?
- state
- start
- action
- update
- stop
- return

How do we do this in scheme (i.e., in a recursive procedure)?

...use a "helper function":

\[
\text{(define (helper [STATE] )}
\begin{align*}
& \text{(cond ( [DONE] [RETURN] )} \\
& \hspace{0.5cm} \text{(else [ACTION]}
\end{align*}
\]

\[
\text{(helper [UPDATE] )})
\]

\[
\text{(define (main) (helper [START] )})
\]
Iteration

To sum ints from a to b  ;;State
State is s  ;;State
Starting with s <- 0  ;;Start
s <- s + a  ;;Update
a <- a + 1  ;;Update
repeat
stop when a > b.  ;;Stop
The answer is the value of s.  ;;Return

(define (sum-helper [STATE])
  (cond ((DONE?) [RETURN])
        (else [ACTION]
               (sum-helper [UPDATE])))
)(define (sum-ints [START] [LAST])
  (define [DONE?] (<= a b))
  (define [ACTION] (set! s (+ s a))
                   (set! a (+ a 1)))
  (define [UPDATE] (+ a 1))
  (define [RETURN] s)
  s)

Consider the following procedures...

(define (our-display x) (display x) x)

(define (count1 x)
  (cond ((= x 0) 0)
        (else (our-display x)
               (count1 (- x 1)))))

(define (count2 x)
  (cond ((= x 0) 0)
        (else (count2 (- x 1))
               (our-display x))))

What happens for each of:
(count1 4)
(count2 4)

are these recursive?

Count up

(define (our-display x) (display x) x)

(define (count2 x)
  (cond ((= x 0) 0)
        (else (count2 (- x 1))
               (our-display x))))

(count2 4)
(count2 3)
(count2 2)
(count2 1)
(print 1)  "1"
(print 2)  "2"
(print 3)  "3"
(print 4)  "4"

Iterative count-up

Write a function count-up-2, but is an iterative process.

(define (count-up-2 x)
  (define (iter [STATE] )
    (cond ((DONE?) #t)
          (else [DO SOMETHING]
                 (iter [UPDATE] ))))
  (iter [START] )
)

Sum-ints iterative

(define sum-ints (lambda (first last)
                        (sum-helper 0 first last)))

(define sum-helper (lambda (sum current last)
                        (cond ((> current last) sum)
                              (else (sum-helper
                                      (+ sum current)
                                      (+ current 1)
                                      last))))))
Iterative count-up

Write a function count-up-2, but is an iterative process.

(define (count-up-2 x)
  (define (iter [STATE] ) ;; THE COUNTER
    (cond ( (DONE??) #t) ;; > x ?
      (else (DO-SOMETHING) ;; Print!
            (iter [UPDATE] )))) ;; Increment!
    (iter [START] ) ;; x MOD by,
       ;; so we end up at x!
  )

Multiply

We can build MUL(A,B) from other primitive ops...

Recursive definition:
  test, base case, recursive case?
  ... add A to mul (A-1, B)...
  ... mul(1,B)=B

Iterative algorithm:
  state start action update stop return?
  ...S=0; for C=B to 1 by -1, do S=S+A; return S

Iterative or Recursive?

(define (mul n m)
  (if (= n 0) 0 (+ (mul (- n 1) m)))))

(define (mul2 n m)
  (define (help count ans)
    (if (= count 0)
      ans
      (help (- count 1) (+ m ans))))
  (help n 0))

(define (mul3 n m)
  (define (help count)
    (if (= count 0)
      0
      (+ m (help (- count 1))))))
  (help n))

Remember...

• calendar....