CS 603: Programming Language Organization

Lecture 13

Spring 2004

Department of Computer Science

University of Alabama

Joel Jones

Outline

- Questions
- μ-Scheme (cont.)

Let's play at the board again (No Books!)

- Property lists–a-list where attribute is an attribute list
- Examples
  - `(set fruits '((apple ((texture crunch)))`
    `(banana ((color yellow)))))`
  - `(getprop 'apple 'texture fruits)`
    `crunchy`
- Write `(getprop x p plist)`, where x is the individual, p is the property and plist is the property list

Let's play at the board again (No Books!)

- `(putprop x p y plist)` give individual x value y for property p in plist
  - `(set fruits (putprop 'apple 'color 'red fruits))`
    `->((apple ((texture crunchy)(color red)))(banana ((color yellow))))`

μ-Scheme

- Closures
  - Pair of lambda (function value) and environment
    - «`(lambda (y) …)`, {x |-> *l*}»
  - Environment maps name to mutable location

```
->(val counter-from
       (lambda (n)
           (lambda () (set n (+ n 1)))))
->(val ten (counter-from 10))
<procedure>
->(ten)
11
->(ten)
12
```

μ-Scheme (cont.)

- Closures

Pair Up:

• Write a function (make-withdraw) that models a bank account balance, where only withdrawals are allowed

```
->(val make-withdraw (lambda (balance) …))

->(val W1 (make-withdraw 100))

->(W1 10)

90
```

Simple higher-order functions

- Composition
  - (define o (f g) (lambda (x) (f (g x))))
  - (define even? (n) (= 0 (mod n 2)))
  - (val odd? (o not even?))

Pair Up:

• Write a function (to8th) using composition with square and ? that raises the input to the 8th power

```
-> (define square(x) (* x x))
square
-> (val to8th …)
<procedure>
-> (to8th 2)
256
```

Higher-order functions on lists

- Filter –

```
(define filter (p? l)
    (if (null? l) '()
        (if (p? (car l))
            (cons (car l) (filter p? (cdr l)))
            (filter p? (cdr l)))))
```

- Exists? –

```
(define exists? (p? l)
    (if (null? l) #f
        (if (p? (car l))
            #t
            (exists? p? (cdr l)))))
```

Higher-order functions on lists (cont.)

- All? –

```
(define all? (p? l)
    (if (null? l) #t
        (if (p? (car l))
            (exists? p? (cdr l))
            #f)))
```

- Map –

```
(define map (f l)
    (if (null? l) '()
        (cons (f (car l)) (map f (cdr l)))))
```

Pair Up:

- How are the arguments of `filter`, `exists?`, and `all?` different from those of `map?`