# CS 603: Programming Language Organization

Lecture 15

Spring 2004

Department of Computer Science

University of Alabama

Joel Jones

# Outline

- Questions
- $\mu$-Scheme (cont.)
- Reading for next time

# Higher-order functions for polymorphism

- Implementing sets
  - Constructing mono-morphic sets requires definition of equality—simple if primitive elements

```
->(val emptyset '())
->(define member? (x s) (exists? ((curry equal? x) s))
->(define add-element (x s) (if (member? X s) s (cons x s)))
->(define union (s1 s2) (foldl add-element s1 s2))
->(define set-from-list (l) (foldl add-element '() l))
```

  - Problem is equality for complex types

# Higher-order functions for polymorphism (cont.)

Pair Up:

- Implement the following set functions for sets of sets:

```
->(define member? (x s eqfun) …)
->(define add-element (x s eqfun) …)
```

# Higher-order functions for polymorphism (cont.)

- How to construct?
  - Add parameter to *every* function
  - Make part of "object" (like C++ virtual function tables)
  - Make list of functions (like C++ templates)

# HOF for Polymorphism: Parameter per Function

```
->(define member? (x s eqfun)
      (exists? ((curry eqfun) x) s))
member?
->(define add-element (x s eqfun)
      (if (member? x s eqfun) s (cons s x)))
add-element
```

So uses will look like:

`(member? X s equal)` or

`(add-element x s =alist?`

# HOF for Polymorphism: Part of "object"

```
->(define mk-set (eqfun elements)
      (cons eqfun elements))
```

Pair Up:

- Draw a diagram (of cons cells and closures) for

```
(mk-set =set? '(a b c))
```

```
->(define eqfun-of (set) (car set))
->(define elements-of (set) (cdr set))
->(val emptyset (lambda (eqfun) (mk-set eqfun '())))
```

# HOF for Polymorphism: Part of "object"

Pair Up:

- What was type of the function "`exists?`" ?

```
(define exists? (p l)…)
```

```
->(define member? (x s)
      (exists? (curry (eqfun-of s)) x)
               (elements-of s)))
->(define add-element? (x s)
      (if (member? X s) s
          (mk-set (eqfun-of s)
                  (cons x (elements-of s)))))
```

# HOF for Polymorphism: Part of "object"

So uses will look like:

```
->(val alist-empty (emptyset =alist?))
->(val s (add-element '((U Thant) (I Ching)) alist-
empty)
(<procedure> ((U Thang) (I Ching)))
->(val s (add-element '((Hello Dolly))))
(<<procedure> ((Hello Dolly) (U Thang) (I Ching)))
->(member? '((I Ching)))
#t
```

Advantage is that only constructor (here, `emptyset`) requires extra parameter

# HOF for Polymorphism: List of Functions

```
->(val mk-set-ops (lambda (eqfun)
       (list2
            (lambda (x s) ; member?
                (exists? ((curry eqfun) x) s))
            (lambda (x s) ; add-element
                (if (exists? ((curry eqfun) x) s)
                    s
                    (cons x s)))))
-> (val list-of-al-ops (mk-set-ops =alist?))
```

Pair Up:

- Draw a diagram (of environment, cons cells and closures) for

```
(val list-of-al-ops (mk-set-ops =alist?))
```

# HOF for Polymorphism: List of Functions

```
-> (val al-member? (car list-of-al-ops))
-> (val al-add-element (cadr list-of-al-ops))
```

So uses will look like:

```
->(val emptyset '())
->(val s (al-add-element '((U Thant)(I Ching))
                         emptyset))
(((U Thant) (I Ching)))
->(val s (al-add-element '((E coli)(I Ching)) s))
(((E coli) (I Ching)) ((U Thang) (I Ching)))
```

# Reading & Questions for Next Class

- Chapter 3.9.2 – 3.10