CS 603: Programming Language Organization

Lecture 14

Spring 2005

Department of Computer Science

University of Alabama

Joel Jones

Outline

- Questions
- μ-Scheme (cont.)
- Reading for next time

Types

- Map: $(\alpha \rightarrow \beta) \times \alpha \text{ list} \rightarrow \beta \text{ list}$
- foldl: $(\alpha \times \beta \rightarrow \beta) \times \beta \times \alpha \text{ list} \rightarrow \beta$

Curry

```
(define curry (f)
  (lambda (x)
    (lambda (y) (f x y))))
(((curry '+) 1) 5)
-> 6
```

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?`

Reading & Questions for Next Class

- Chapter 3.9.2 – 3.10