CS 313 Lecture 23

Scheme wrap-up
Evaluation order
Intro to ML
Scheme wrap-up

Common functional tools

• map
  • apply a function to each element in a list
  • [1, 2, 3, 5, 7, 8] add 10 \(\rightarrow\) [11, 12, 13, 15, 17, 18]

• filter
  • sublist of elements in a list satisfying predicate
  • [1, 2, 3, 5, 7, 8] odd? \(\rightarrow\) [1, 3, 5, 7]

• reduce / fold
  • combine all elements in a list into a single value
  • [1, 2, 3, 5, 7, 8] add \(\rightarrow\) 1 + 2 + 3 + 5 + 7 + 8
map, filter, foldl / foldr

(map fun list)
(map (lambda (x) (+ x 10)) '(1 2 3 5 7 8))

(filter pred list)
(filter odd? '(1 2 3 5 7 8))

(foldl fun init list)
(foldl + 0 '(1 2 3 5 7 8))
Evaluation order

• What parameter-passing mechanism does Scheme use?
  • call by value?
  • call by reference?
  • call by value-result?

→ all equivalent in pure functional language (no assignment)

• But:
  • call by value
  • call by name

results can differ in some scenarios
Two approaches to expression evaluation (Sethi 8.4)

**Innermost**
- call by value
- application-order evaluation
- eager evaluation
- evaluate args, then run function body

**Outermost**
- call by name
- normal-order evaluation
- lazy evaluation
- substitute args into function body, evaluate as needed

• Scheme, ML
• Haskell
Intro to ML

• Functional language, like Scheme, but *typed*
  • *infers* types in most situations

• Like with Scheme, we will focus on “pure” part of language

• We’ll use SML/NJ (Standard ML of New Jersey)
  • installation instructions are posted

• Basic types:
  int   real   bool   char   string