Concept of Programming Languages (CS320)  
Lecture 6

By Zhiqiang Ren (Alex)  
aren@cs.bu.edu
Content

- High Order Function revisited
- Closure revisited
- How to write lambda
- Quiz
Higher Order Function

• Glue the computation pattern with parameters (including functions)

```plaintext
fun sum_accu(xs: list0 int, accu: int): int =
case+ xs of
| cons(x, tail) => let
  val accu' = sum(x, accu)
in
  sum_accu (tail, accu')
end
| nil () => accu

fun f_accu(xs: list0 int, accu: int, foo: (int, int) -> int): int =
case+ xs of
| cons(x, tail) => let
  val accu' = foo(x, accu)
in
  sum_accu (tail, accu')
end
| nil () => accu
```
Application of High Order Function

- Iterate
- Map
  - Matrix transpose
- Filter
  - Remove duplicate

// [remove_dup] removes all the duplicates in a given list. E.g, // the list [1,7,2,1,3,5,3] becomes [1,7,2,3,5] after all duplications // are removed.
Closure Revisited

- closure = function with extra data

```haskell
fun createClosure (x: int): int -<cloref1> int = let
  fun plusX (y: int):<cloref1> int = x + y
in
  plusX
end

val plus3= createClosure(3)
val ret = plus3 (1)
```

- Obj Oriented style
lambda expression

• lambda expression is closure without name

• Syntax

// Without type
val foo1 = lam (x, y) => x + y  // may cause compiling error

// With just types for parameters
val foo2 = lam (x: int, y: int) => x + y

// With full types
val foo3 = lam (x: int, y: int) => (x + y): int

val ret = foo2 (1, 2)
val ret = (lam (x: int, y: int) => x + y) (1, 2)

val foo4 = lam (t: (int, int)) => t.0 + t.1
val ret = foo4 @(1, 2)
Summary of syntax

• Type (signature) of closure

fun foo (x1: tp1, x2: tp2): tp3
fun foo (x1: tp1, x2: tp2):<fun1> tp3
fun foo (x1: tp1, x2: tp2):<cloref1> tp3

lam (x1: tp1, x2: tp2)=> (body): tp3
lam (x1: tp1, x2: tp2)=<fun1> (body): tp3
lam (x1: tp1, x2: tp2)=<cloref1> (body): tp3

(tp1, tp2) -> tp3
(tp1, tp2) -<fun1> tp3
(tp1, tp2) -<cloref1> tp3
How to write lambda

- \( \text{fun\{a,b:t@ype\} A (f: (a, a) -<\text{clorefi}> b): a -<\text{clorefi}> b} \)

- \( \text{fun\{a,b,c:t@ype\} B (f: (a, b) -<\text{clorefi}> c): (b, a) -<\text{clorefi}> c} \)

- \( \text{fun\{a,b,c:t@ype\}} \)
  \( \text{compose (f: a -<\text{clorefi}> b, g: b -<\text{clorefi}> c): a -<\text{clorefi}> c} \)

- \( \text{fun\{a,b,c:t@ype\} CP} \)
  \( \text{CP (f: a -<\text{clorefi}> b, g: (a -<\text{clorefi}> c) -<\text{clorefi}> c, h: b -<\text{clorefi}> c): c} \)
Quiz

• Turn the following function into tail recursive version.

```plaintext
// assume input >= 0
fun foo (x: int): int =
  if x = 0 then 0
  else if x = 1 then 1
  else if x = 2 then 2
  else foo (x - 1) + foo (x - 2) + foo (x - 3)
```