Concept of Programming Languages (CS320)
Lecture 11

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

- Continuation Revisited
- Lazy Evaluation (Stream)
Continuation

- The recursive call has to be the last step.

// direct style
fun f91d(x: int): int =  
  if x > 100 then x - 10 
  else 
    f91d (f91d (x + 11)) // end of [f91d]

// continuation-pass style

abst@ype ans // checking cps

fun kf91el (x: int, k: int -<cloref1> ans): ans = 
  if x <= 100 then kf91el (kf91el (x+11, k), k) 
  else k(x) // end of [kf91el]

fun kf91 (x: int, k: int -<cloref1> ans): ans = 
  if x > 100 then k (x - 10) 
  else 
    kf91(x + 11, lam x => kf91 (x, k)) // end of [kf91]

assume ans = int
Continuation

- Tree Traversal

   // Wrong version

   fun traverse_mtree (t: mbtree): mlist = let
       abstype ans
       fun loop (t: mbtree, k: mlist -<cloref1> ans): ans =
         case+ t of
         | mbtree_cons (node, t1, t2) =>
           loop (t2, lam (xs: mlist): mlist =>
             k (mlist_cons (node,
               loop (t1, lam (ys: mlist): ans => append (ys, xs))
             )
             )
           )
         | mbtree_nil () => k (mlist_nil)

         assume ans = mlist
         in
         loop (t, lam xs => xs)
   end
Lazy Evaluation

- Is “1 + 1” equal to “2”? 
- **lazy** is type constructor supported by ATS, $delay$ is the corresponding constructor
  - 2 is of type int
  - $delay(1 + 1)$ is of type lazy (int)

- The usage of !
  - val $delay (x) = lazyvalue$
- Usage of $effmask_all()$
  - Delayed Effect is evil. (overdrawing your checking account)
Stream

- **prelude/basics_stata.sats**

```haskell
abstype lazy_t0ype_type (t@ype+) // boxed type // unnamed
stadef lazy = lazy_t0ype_type

datatype stream_con (a:t@ype+) =
| stream_nil (a)
| stream_cons (a) of (a, stream a)

where stream (a:t@ype) = lazy (stream_con a)
```
Diagonalization

- Euler’s transform
Stream of directory entry

- stream (direntry)