CAS CS 320
Lec 11

Continuation

1. Concept
    a. val _ = foo(x, _)
    When the result of foo(x) is computed, send the return value to the _ next to it
    b. fun fact(x) =
        if x > 0 then x * fact(x-1) else 1
    c. fun kfact (x,k) (*continuation passing style*) =
        k(fact(x))
    d. fun kfact(x,k) =
        if x > 0 then kfact(x-1, fn res => k(x*res))
        (*tail recursive (can be turned into loop and implement it without using loop *)
        (* need space to create closure fn res => k(x * res) → stored on heap *)
        (* heap is much larger than stack *)
        else k(1)
    e. Steps
        i. kfact(2, fn res => res) => 2
        ii. kfact(1, fn res => (2 * res) => 2
        iii. kfact(0, fn res => (1 * res)) => 1
    f. fun fibo(x) =
        if x <=1 then x else fibo(x-2) + fibo(x-1)
    g. fun kfibo(x, k) =
        if x <= 1 then k(x)
        else kfibo(x-2, fn res1 => kfibo(x-1, fn res2 => k(res1+res2))
    h. fun kfact2(x,k) =
        if x > 0 then kfact(x-1, fn res => k(x * res))
        else k(k(1))
    kfact2(3, fn res => res) = 1 * 2 * 3 * [1 * 2 * 3 * [1]] = 6 * 6 = 36
    (* must be tail recursive when using continuation *)
    i. fn f91(x) =
        if x > 100 then x - 10
        else f91(f91(x+11))
    j. fun kf91(x,k) =
        if x > 100 then k(x-10)
        else kf91(x+11, fn res => kf91(res, k))

k. fun f(x) = f(x+1)
   f(0) ⇒ will reach/cause overflow in SML, but it is tail recursive

l. fun f(x) =
   ```
   if x > 100 then f(0)
   else f(x+1)
   ```
   Execute forever, tail recursive

m. fun iseven(x) =
   ```
   if x > 0 then isodd(x-1)
   else true
   ```
   and
   fun isodd(x) =
   ```
   if x > 0 then iseven(x-1)
   else false
   ```
   (* mutually tail recursive call *)

n. fun kiseven(x,k) =
   ```
   if x > 0 then kisodd(x-1)
   else k(true)
   ```
   and
   fun kisodd(x,k) =
   ```
   if x > 0 then kiseven(x-1)
   else k(false)
   ```

2. Time Complexity
   a. if length(xs) = 0 then → O(n) to find length of list
   b. In python, xs.append(y) → O(1) and xs.insert(0,x) → O(n) (use less insert)
   c. In SML, putting something on the very front is O(1), at the end is O(n)
   d. if f is O(n) then f is O(n^2) → true
      because O(n) is bounded by O(n^2) (it is upper bound)
3. Recurrence (relation/equation)
   a. fun fact(x) =
      ```
      if x > 0 then x * fact(x-1) else 1
      ```
   b. T(n) = T(n-1) + O(1)
      …
      T(0) = O(1)
      → T(n) = n * O(1) = O(n)
      T(n) is O(n)
   c. fun fibo(x) =
      ```
      if x < 2 then x else fibo(x-2) + fibo(x-1)
      ```
   d. T(n) = T(n-1) + T(n-2) + O(1)
      <= 2 * T(n-1) + O(1)
      T(n) is O(2^n)

e. Time complexity for continuation style and direct style is the same
f. Mergesort → O(n*log (n)) → still practical
g. Quicksort → O(n^2)