Lec 6 : COL765

Agenda

- Revisit tail-recursion
  - ↳ discuss that you require invariants
- Revist recursion
  - ↳ discuss that you require inductive step
- finish searching from yesterday!
- Introduce higher order functions!

Recap

- Lists, tuples,
- Sorting & Searching

Recursion:

Problem solution described in terms of itself

What is required is the base case, I.H. & I.S.

→ tail recursion: you additionally model for the state of computation at each step of function invocation

- also, tail recursion is the last step in the computation of the function.

+ Additionally, one requires an invariant on the state of computation & shows through induction the invariant is indeed true.

+ Once invariant is shown to be correct, then use substitution to show that tail recursive process computes the desired result.

Eg : Fastpow → tail recursive

$$\text{fastpow } x\ n = \begin{cases} 1 & \text{if } n = 0 \\ \text{fastpow } (x \cdot x)\ (n \text{ div } 2) & \text{if } n > 0 \wedge n \text{ is even} \\ x \cdot \text{fastpow } (x \cdot x)\ (n \text{ div } 2) & \text{if } n > 0 \wedge n \text{ is odd} \end{cases}$$

Correctness

- induct on n
- B: $x^0 = 1 =$ fastpow x 0
- I.H: $0 \le m < n$ fastpow $x\ m = x^m$ [Complete induction]
- I-S: $n > 0 \rightarrow$ odd: $n = 2k+1$ ∴ $n \text{ div } 2 = k$

$x \cdot \underbrace{\text{fastpow } x^2\ (2k+1)/2}_{\text{I.H.}} = x \cdot (x^2)^{n \text{div} 2} = x \cdot (x^2)^k = x\, x^{2k} = x \cdot x^{n-1} = x^n$

Efficiency

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

$T(n) = m + 1$

$= \log_2 n + 1$

$= O(\log_2 n)$

Fastpow tail recursive

- Auxillary var p to hold the o/p of
  Step i.
- fastpow_tl y m p =

$$\begin{cases} p & \text{if } m = 0 \\ \text{fastpow\_tl } y^2 \ (m \text{ div } 2) \ p & \text{if } m \text{ is even} \\ \text{fastpow\_tl } y^2 \ (m \text{ div } 2) \ (p \cdot y) & \text{otherwise} \end{cases}$$

Correctness Invariant

$\forall\ y, p > 0,\ m \geq 0$

fastpow_tl y m p $= p \cdot y^m$

Proof by Induction: Base: fastpow_tl y 0 p = p $= p \cdot y^0$

I.H.: $0 \leq k < m$ fastpow_tl y k p $= p \cdot y^k$

I.S.: $m > 0 \longrightarrow$ Odd case, $m = 2j+1$

Complete induction!

fastpow_tl y m p = fastpow_tl y^2 (2j+1) div 2 (y·p)

By. I.H. $\equiv (y \cdot p)(y^2)^{(2j+1 \text{ div } 2)}$

$= (y \cdot p)(y^{2j})$

$= p \cdot y^{2j+1} = p \cdot y^m$

Searching with lists

Linear search

```
search x ls =
    case ls of
        []     -> false
        (y:ys) -> if x == y then True
                  else search x ys
```

Bin. Search

List with indices !! → The operator
you would count to use

Higher Order functions

- Till now we have seen functions which accept inputs of primitive types and return numbers as output

- Exploit the observation: many computations are similar in structure

  Eg: sum, sum of squares, ...

  can all be reduced to

$$\sum_{x=l}^{u} f(x)$$

For sum
$f(x) = x$

Sum of squares
$f(x) = x^2$ ....

Integral of smooth functions etc.

— Clearly due to this commonality in computation structure, a generic summation can be defined!

Things to note:

- three important components in the general summation fnc.
  (1) lower & upperbound for summation range
  (2) The arbitrary function 'f' [also called the term function]
  (3) The successor function indicating how lower bound steps up towards the upper bound!

It is NOT advisable to define summation function with f and succ as free parameters, because then summation function can't be used as a black box.

f & succ should be defined with global scopes with the same names

Polymorphic functions

→ a slight detour

There was still something unsatisfactory about our summation construction.

Why? Observe $\prod_{i=l}^{u} f(i)$

This also has a similar computational structure as summation except + is replaced with *

Higher order list functions

- maps : applies f uniformly to a list
- filters : take a predicate & returns a sublist of elements for which the predicate is true.

filter pred ls =
$$\begin{cases} [] & \text{if } ls = [] \\ (\text{head } ls) : (\text{filter pred } (\text{tail } ls)) & \text{if pred (head ls)} \\ \text{filter pred (tail ls)} & \text{otherwise} \end{cases}$$

Eg: choosing evens or odds from the list.

pred x = if x mod 2 == 0 then True else False

positives ls = filter pos ls

where pos x = x > 0

Next class foldl & foldr