COMP90048 Declarative Programming
Semester 1, 2018
Peter J. Stuckey

Declarative Programming

Workshop exercises set 11.

QUESTION 1
Write a function fibs :: Int -> [Integer] which returns a list containing the first n numbers in the Fibonacci sequence: [0,1,1,2,3,5,8,...], where the third and subsequent numbers are the sum of the two preceeding numbers (0+1=1, 1+1=2, 1+2=3, 2+3=5, etc). We use Integer rather than Int because the numbers grow exponentially and therefore overflow native Ints quite quickly. Is the algorithmic complexity of your solution acceptable?

QUESTION 2
If we do pairwise addition of the elements of the Fibonacci sequence and its tail, we get the tail of the tail of the sequence:

```
  0 1 1 2 3 5 8 ...     fibs
+ 1 1 2 3 5 8 ...       tail fibs
= 1 2 3 5 8 ...         tail (tail fibs)
```

Use this property to write a definition of allfibs :: [Integer] which is the (infinite) Fibonacci sequence (Hint: the zipWith Prelude function is useful). Define fibs in terms of allfibs. How efficient is this definition of fibs compared to your previous one?

QUESTION 3
Consider the bottom-up merge sort implementation from workshop 2.

```
>mergesort xs = repeat_merge_all (merge_consec (to_single_els xs))
>
>to_single_els [] = []
>to_single_els (x:xs) = [x] : to_single_els xs
>
>merge [] ys = ys
>merge (x:xs) [] = x:xs
>merge (x:xs) (y:ys)
>       | x <= y = x : merge xs (y:ys)
>       | x >  y = y : merge (x:xs) ys
>
>merge_consec [] = []
>merge_consec [xs] = [xs]
>merge_consec (xs1:xs2:xss) = (merge xs1 xs2) : merge_consec xss
>
>repeat_merge_all [] = []
>repeat_merge_all [xs] = xs
>repeat_merge_all xss@(_:_:_) = repeat_merge_all (merge_consec xss)
```

With list xs of length n, what is the maximum additional space that is needed at any one time, assuming strict evaluation, for evaluating merge_consec (to_single_els xs)? What if lazy evaluation is used instead?

What is the maximum additional space is needed at any one time, assuming strict evaluation, for evaluating mergesort xs? Can we do significantly better than this?

QUESTION 4
Consider an interpreter for a language which produces a pair

containing the result of the computation plus some debugging information, which is a string containing information about all assignment statements and function calls. Compare the efficiency of the following:

a) Execution of the interpreter using strict evaluation and printing the debugging string.
b) Execution of the interpreter using lazy evaluation and printing the debugging string.
c) Execution of the interpreter using strict evaluation but not printing the debugging string.
d) Execution of the interpreter using lazy evaluation but not printing the debugging string.
e) Execution of a similar interpreter which doesn't produce the debugging string at all.

QUESTION 5
Here are some students' answers to one of the questions on a sample mid-semester test, which were posted on the LMS (thanks to the authors). The question asked for a Haskell function to print out Mtrees with indentation showing the structure. Compare and contrast these solutions. Can you come up with something better than all three?

```
>data Mtree a = Mnode a [Mtree a]

>print_mtree :: Show a => Mtree a -> IO()
>print_mtree tree = indent_mtree 0 tree
>    where
>        indent_mtree :: Show a => Int -> Mtree a -> IO()
>        indent_mtree i (Mnode val children) = do
>            putStrLn $ (replicate i ' ') ++ (show val)
>            foldl (>>) (return ()) (map (indent_mtree (i+1)) children)

>type Line = String
>
>print_mtree' :: Show a => Mtree a -> IO ()
>print_mtree' t =
>    let
>        toLines :: Show a => Mtree a -> [Line]
>        toLines (Mnode val cs) = show val : map (' ':) (concatMap (toLines) cs)
>    in  foldl (\acc str -> acc >> (putStrLn str)) (return ()) (toLines t)
>
>-- A clearer version
>print_mtree2 :: Show a => Mtree a ->  IO ()
>print_mtree2 t =
>    let
>        toLines :: Show a => Mtree a -> [IO ()]
>        toLines (Mnode val cs) =
>            print val : map (putChar ' ' >>) (concatMap (toLines) cs)
>    in  foldl1 (>>) (toLines t)
```