CPSC 449 Fall 2020

Assignment #2 Written Part

Ali Akbari

30010402

Tutorial #3

1 - 2 Coding Portion is in A2.hs

3. a)[10%]Prove that the implementation of merge sort in 2(c) always terminates.

The merge sort implementation in our code uses two other functions splitlist and mergelists.

splitlist splits a list in two and puts the elements in each list based on the indices if they are odd or even. Even indices elements in the first list and odd indices elements in the second list. The split function terminates as the recursive call is on the rest of the list once the head is computed and place in the right list. So the length of the list is reduced by one each recursive call, if rank function maps to length of list of size n then in each recursive when one element is placed in the right list then it be n - 1. n - 1 < n, so it is decreasing as the recursion unfolds and reaches the base case. Thus, this function terminates.

mergelists merges two list into one and puts the elements from each list in ascending order based on the value of each element. The merge function terminates as the recursive call is on the rest of the list once it is computed for the head and it in the right place in the list. So the length of the list is reduced by one each recursive call, as one element is sorted each time. If rank function maps to length of list of size n then in each recursive when one element is placed in the right position then it will be n - 1 elements left. n - 1 < n, so it is decreasing as the recursion unfolds and reaches the base case. Thus, this function terminates.

To prove for termination for mSort the following rank function is defined. The rank function maps to the total length of the input list to be merge sorted. Suppose the 'wholelist' is of size n. Then with the two halves after the split in the recursion calls makes the rank input as $\frac{n}{2}$ if the halves are even and $\frac{n+1}{2}$ if it is odd. Which is strictly smaller than the length of the list n, so as the rank of the argument decreases in the recursion, the function mSort is guaranteed termination as the recursion unfolds and reaches the base case of either one element or empty list.

b) Consider the following function

Consider the following function.

```
mystery :: [[Integer]] -> Integer
mystery [] = 0
mystery ((x : xs) : ys) = x + mystery (xs : ys)
mystery ([] : ys)     = mystery ys
```

i.[5%] Give a conjecture of what the mystery function returns.

The function mystery takes in a list of lists and returns the
total addition of each integer element from the inner lists. The function runs through each element of the lists recursively and adds them up, base case of an empty list for recursion return 0.

ii.[10%] Prove that the mystery function terminates for all inputs

To prove for termination the following rank function is defined. The rank function maps to the total length of each inner list summed, in the outer list. Let the list of lists be ((x:xs):ys). Let the length of the whole list be n and the length of the inner lists be m, ((x:xs):ys) = n = ($1 + (m_2 + m_3 + m_4 + ...m_x)$). ($1 + (m_2 + m_3 + m_4 + ...m_x)$) is the input rank. Recursive Call #1: The argument of the recursive call is mystery (xs:ys), therefore the input rank will be $(m_2 + m_3 + m_4 + ...m_x)$ which means the length decreasing by 1 as the head of the list is removed. Recursive Call #2: The argument of the recursive call is mystery ([]:ys), therefore the input rank will be $1 + (m_2 + m_3 + m_4 + ...m_x)$ which means the length has not changed as empty lists have a length of 0, therefore rest of the list is sent and the first recursive guard will take it in and based on the above the length of the whole list should decrease. Rank of the argument decreases strictly as recursion unfolds. The function mystery is guaranteed to terminate.

4. Prove, by structural induction, for all finite lists xs, we have:

length xs = length (reverse xs) **(*)**

You may assume the following equations for reverse and length:

Definitions

reverse [] = []	(reverse.1)
reverse (x:xs) = (reverse xs) ++ [x]	(reverse.2)
length [] = 0	(length.1)
length (x:xs) = 1 + (length xs)	(length.2)
length (xs ++ ys) = (length xs) + (length ys)	(length.3)

a. **Proof Goals:**

We want to prove that for all finite lists:

First, we are going to prove the base case:

length [] = length(reverse []) (BASE)

Then, we are going to prove the induction step:

length(x:xs)= length(reverse (x:xs)) (IND)

Assuming the hypothesis:

length xs = length (reverse xs) (HYP)

b. **Proving the base case:**

We have to prove that length [] = length(reverse []) (BASE)

Let xs = []

Then,

Left-hand side:

length([]) = 0 by length.1

Right-hand side:

reverse [] = [] by reverse.1

length(reverse []) = length([]) by reverse.1

length([]) = 0 by length.1

Left-hand side = Right-hand side

c. Proving the Induction step:

length(x:xs)= length(reverse (x:xs))	(IND)
length xs = length(reverse xs)	(HYP)

Left-hand side:

length(x:xs) =	
length(x:xs) = 1 + (length xs)	by length.2
1 + (length xs)	

Right-hand side:

length(reverse (x:xs))	
= length((reverse xs) ++ [x])	by reverse.2
= length (reverse xs) + length [x]	by length.3
= length(reverse xs) + length x:[]	by list construction
= length(reverse xs) + 1 + (length [])	by length.2
= length(reverse xs) + 1 + 0	by length.1
= 1 + length(reverse xs)	by Math
= 1 + length xs	by HYP

Left-hand side = Right-hand side

End of Proof.