CPSC 449 Fall 2020

Final Written Part

Ali Akbari

30010402

Tutorial #3

Question #3

```
crunch :: [Integer] -> [Integer] -> Integer
crunch [] []     = 0                                     (crunch.1)
crunch (x:xs) [] = crunch xs [x]                         (crunch.2)
crunch [] (y:ys) = y + crunch [] ys                      (crunch.3)
crunch (x:xs) (y:ys)
  | even x       = crunch xs (x:y:ys)                    (crunch.4)
  | otherwise    = crunch ((x+y):xs) ys                  (crunch.5)
```

a) The function crunch takes in two lists of integers and returns the integer total addition/sum of all integers from both the lists. The function runs through each element of the twol lists recursively and makes a single list to sum it up, base case of an empty list for recursion return 0.

b) To prove for termination the following rank function rank(xs;ys) = n, is defined. The rank function maps to the total length of both list lengths summed up. Let the two lists be(xs),(ys). Let the total length of both lists be n where n = L1 + L2 (L1 = length of first list(x:xs) and L2 = length of first list(y:ys)).

c) The crunch function has one base case, where both lists are either recursively called with empty lists or when they are originally passed in as empty. We have to keep in mind these base cases when determining the termination of the crunch function. As previously defined the rank function is the length of the two lists summed, n.

When the algorithm reaches the base case after completing recursive call #2, (crunch.3) the total length of the both list becomes 0 and reaches base case #1(crunch. 1). Each recursive call for (crunch.3) removes the head and adds it to a sum, thus decreasing the length by 1 until empty (decreasing n by 1, n - 1 = L1 + L2 - 1). Each recursive call for (crunch.4) merges list one into list two until list one is empty, thus calling back to the recursive call (crunch.3) as first list is empty and second one is not and also decreasing the length by 1 until empty by (crunch.3) as shown above. Each recursive call for (crunch.5) adds the head of list two to the head of list one until the list

two is empty, thus decreasing the total length by one, this does alter the length of the second list as the head(y) is used every iteration, thus decreasing n by 1, n - 1 = L1 + L2 - 1. This further falls into recursive call #1 (crunch.2) then (crunch.2) puts list 1 element into the second which fall into recursive call #2 which we already saw decrease the length by 1 until empty.
This shows that the Rank rank(xs;ys) = n of the argument decreases strictly as the recursion unfolds.
The function crunch is guaranteed to terminate.

Question #5

```
Definitions:
Type VarName = Char

Data Expr = Lit Integer
          | Var VarName
          | Add Expr Expr

numVars :: Expr -> Integer
numVars (Lit _) = 0                                    (numVars.1)
numVars (Var _) = 1                                    (numVars.2)
numVars (Add e1 e2) = (numVars e1) + (numVars e2)      (numVars.3)

leftHeight :: Expr -> Integer
leftHeight (Lit _) = 0                                 (leftHeight.1)
leftHeight (Var _) = 0                                 (leftHeight.2)
leftHeight (Add e1 _) = 1 + (leftHeight e1)            (leftHeight.3)

size :: Expr -> Integer
size (Lit _) = 1                                       (size.1)
size (Var _) = 1                                       (size.2)
size (Add e1 e2) = 1 + (size e1) + (size e2)           (size.3)
```

Use structural induction to prove that, for every finite expression e,
(numVars e)+(leftHeight e) ≤ size e

a) **Principle of Structural Induction for algebraic type Expr:**
To prove that E(e) holds for all finite expression e prove the following:
1) E(Lit n)
2) E(Var v)
3) E(e1) and E(e2) $\Rightarrow$ E(Add e1 e2)

Proof Goals:
1) (numVars (Lit n))+(leftHeight (Lit n)) $\leq$ size (Lit n) (Base.1)
2) (numVars (Var v))+(leftHeight (Var v)) $\leq$ size (Var v) (Base.2)

Assume:
3) (numVars e1)+(leftHeight e1) $\leq$ size e1 (hyp.1)
4) (numVars e2)+(leftHeight e2) $\leq$ size e2 (hyp.2)

Prove:
5) (numVars (Add e1 e2))+(leftHeight (Add e1 _)) $\leq$ size (Add e1 e2)(Ind.1)

b) **Base Case:**
Want:
(numVars (Lit n))+(leftHeight (Lit n)) $\leq$ size (Lit n) (Base.1)
(numVars (Var v))+(leftHeight (Var v)) $\leq$ size (Var v) (Base.2)

1) (numVars (Lit n))+(leftHeight (Lit v)) $\leq$ size (Lit n) (Base.1)
LHS:
numVars (Lit n) = 0 (numVars.1)
leftHeight (Lit n) = 0 (leftHeight.1)
(numVars (Lit n))+(leftHeight (Lit n))
0 + 0 = 0 (by addition)
RHS:
size (Lit n) = 1 (size.1)
0 $\leq$ 1
LHS $\leq$ RHS
So base case holds for:
(numVars (Lit n))+(leftHeight (Lit n)) $\leq$ size (Lit n) (Base.1)

2) (numVars (Var v))+(leftHeight (Var v)) $\leq$ size (Var v) (Base.2)
LHS:
numVars (Var v) = 1 (numVars.2)
leftHeight (Var v) = 0 (leftHeight.2)
(numVars (Var v))+(leftHeight (Var v))
0 + 1 = 1 (by addition)
RHS:
size (Var v) = 1 (size.2)
1 $\leq$ 1
LHS $\leq$ RHS
So base case holds for:
(numVars (Var v))+(leftHeight (Var v)) $\leq$ size (Var v) (Base.2)

c) Induction Step:

Want:

(numVars (Add e1 e2))+(leftHeight (Add e1 e2)) ≤ size (Add e1 e2)(Ind.1)

Assume:

(numVars e1)+(leftHeight e1) ≤ size e1 (hyp.1)

(numVars e2)+(leftHeight e2) ≤ size e2 (hyp.2)

LHS:

(numVars (Add e1 e2))+(leftHeight (Add e1 e2))

(numVars e1) + (numVars e2) + (leftHeight (Add e1 _)) (numVars.3)

(numVars e1) + (numVars e2) + 1 + (leftHeight e1) (leftHeight.3)

RHS:

size (Add e1 e2)

1 + (size e1) + (size e2) (size.3)

No further simplification can be made but using our assumption in our inductive hypothesis we can still prove the induction step.

(numVars e1) + (numVars e2) + 1 + (leftHeight e1) ≤ 1 + (size e1) + (size e2)

We see that we can remove the 1 (like terms) from either side without changing the inequality.

So we get this,

(numVars e1) + (numVars e2) + (leftHeight e1) ≤ (size e1) + (size e2)

1st Half (numVars e1)+(leftHeight e1) ≤ size e1:

From (hyp.1) we know that

(numVars e1)+(leftHeight e1) ≤ size e1

2nd Half (numVars e2)≤ (size e1) + (size e2):

From (hyp.2) we know that

(numVars e2)+(leftHeight e2) ≤ size e2

In our inductive step we only have

(numVars e2)

but it must be true that if we take away **(leftHeight e2)** from

(numVars e2) + (leftHeight e2)

It must become smaller as **(leftHeight e2)** only returns non-negative numbers, since the recursive steps adds a 1 and base case does not get passed 0, thus **(leftHeight e2)** > 0 so,

(numVars e2) ≤ size e2

If **(numVars e1)+(leftHeight e1) ≤ size e1** from first half and

(numVars e2) ≤ **size e2** then puting both halves together it still holds:

(numVars e1) + (numVars e2) + 1 + (leftHeight e1) ≤ 1 + (size e1) + (size e2)

LHS ≤ RHS .