CPSC 449 — Principles of Programming Languages
Theory Components of Final Exam

Name: Haohu Shen
Student ID: 30063099
Tutorial: 03

Question 3

Consider the following function.

```
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)
```

(The source code above can also be found in the file *SampleCode.hs*.)

(a) Give a conjecture of what the *crunch* function returns. **Hint:** Use equational reasoning to help yourself understand how recursion unfolds in *crunch*. There is, however, no need to include such equational reasoning work with your submission. Only report your conjecture.

Solution. The *crunch* function takes two lists of integers and returns their sum.

(b) Propose a rank function that can be used for demonstrating the termination of *crunch*. More specifically, formulate a function $rank(xs, ys)$ that takes two lists of $Integers$ as input, and returns a natural number as output.

Solution. We propose a rank function that takes two lists of $Integers$ xs and ys and which maps the input lists to a natural number. Suppose the length of xs is m, the length of ys is n, then:

$$\text{rank(xs, ys)} = \text{m+n} \qquad \text{(rank.1)}$$

(c) Use the rank function you formulated above to prove that *crunch* always terminates when given finite lists as input.

Proof. In order to prove the *crunch* always terminates when given finite lists as input, we prove that there is a strict decrease of rank when recursion occurs in each recursive equation.

We firstly look at the equation *crunch*.3. The equation involves 1 recursion call. Thus we prove that $rank([\,], y : ys) > rank([\,], ys)$:

```
rank ([],y:ys)
= 0 + (1 + n)                                   by (rank.1)
= 1 + n                                         by arith.
> n                                             by arith.
```

And

```
rank ([],ys)
= 0 + n                                                    by (rank.1)
= n                                                        by arith.
```

Thus $rank([\,], y : ys) > rank([\,], ys)$ by replacement. Secondly we look at the equation *crunch*.5. The equation involves 1 recursion call. Thus we prove that $rank(x : xs, y : ys) > rank((x + y) : xs, ys)$.

```
rank (x:xs, y:ys)
= 1 + m + 1 + n                                            by (rank.1)
> 1 + m + n                                                by arith.
```

And

```
rank((x+y):xs, ys)
= 1 + m + n                                                by (rank.1)
```

Thus $rank(x : xs, y : ys) > rank((x + y) : xs, ys)$ by replacement.

Thirdly we look at the equation *crunch*.4. The equation involves 1 recursion call. Since the rank function in this step does not change, we consider where is its next stop, firstly, it may repeat the execution of the *crunch*.4 again, but the length of first list is decreased by one, thus after finite operations, it must execute *crunch*.5 or *crunch*.3 if the first list is empty. Since we have proved the rank function is strictly decreased in these two branches, we can say the rank function is strictly decreased in the recursion call in *crunch*.4.

Finally we look at the equation *crunch*.2. The equation involves 1 recursion call. Since the rank function in this step does not change, we consider where is its next stop, since both of its lists are not empty, it must execute the *crunch*.4 or *crunch*.5. Since we have proved the rank function is strictly decreased in these two branches, we can say the rank function is strictly decreased in the recursion call in *crunch*.2.

Since we have proved that there is a strictly decrease of rank when recursion occurs in each recursive equation, we can conclude that the *crunch* always terminates when given finite lists as inputs.

Question 5

Recall the simple expression language above.

```
type VarName = Char
data Expr = Lit Integer
          | Var VarName
          | Add Expr Expr
```

The function $numVars$ returns the number of variable occurrences in a given expression.

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

The function $leftHeight$ returns the number of *Add* appearing in the leftmost branch of a given expression.

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

The function $size$ returns the number of constructors used for constructing a given expression.

```
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,

$$\text{(numVars e)} + \text{(leftHeight e)} \leq \text{size e}$$

The presentation of your proof shall conform to the following format. Deviation will lead to significant mark reduction.

(a) State the Principle of Structural Induction for the algebraic type $Expr$.
Solution. From the description above we have the theorem P(e) as

$$\text{(numVars e)} + \text{(leftHeight e)} \leq \text{size e}$$

Thus to prove that P(e) holds for all finite expression e, prove the following:

i. P(Lit _)
ii. P(Var _)
iii. $P(\text{e1}) \wedge P(\text{e2}) \implies P(\text{Add e1 e2})$

(b) State the concrete proof goals.
Solution. The proof goals are:

- **Base Cases**

$$\text{(numVars (Lit \_))} + \text{(leftHeight (Lit \_))} \leq \text{size (Lit \_)} \qquad \text{(base.1)}$$
$$\text{(numVars (Var \_))} + \text{(leftHeight (Var \_))} \leq \text{size (Var \_)} \qquad \text{(base.2)}$$

- **Induction Steps**
  - **Assume:**

$$\text{(numVars e1)} + \text{(leftHeight e1)} \leq \text{size e1} \qquad \text{(hyp.1)}$$
$$\text{(numVars e2)} + \text{(leftHeight e2)} \leq \text{size e2} \qquad \text{(hyp.2)}$$

– **Prove:**

$$(\text{numVars (Add e1 e2)}) + (\text{leftHeight (Add e1 e2)}) \leq \text{size (Add e1 e2)} \quad \text{(ind.1)}$$

(c) Prove the proof goals.

Solution:

Firstly, we prove the first base case:

- **Want:**

$$(\text{numVars (Lit \_)}) + (\text{leftHeight (Lit \_)}) \leq \text{size (Lit \_)}$$

- **Left-hand side:**

$$\begin{aligned}(\text{numVars (Lit \_)}) + (\text{leftHeight (Lit \_)}) &= 0 + (\text{leftHeight (Lit \_)}) && \text{(by numVars.1)}\\ &= 0 + 0 && \text{(by leftHeight.1)}\\ &= 0 && \text{(by arith.)}\end{aligned}$$

- **Right-hand side:**

$$\begin{aligned}\text{size (Lit \_)} &= 1 && \text{(by size.1)}\\ &\geq 0 && \text{(by arith.)}\\ &\geq \text{L.H.S} && \text{(by replacement)}\end{aligned}$$

Thus it shows that the left-hand-side is smaller than or equal to the right-hand-side, which completes the proof of the first base case.

Secondly, we prove the second base case:

- **Want:**

$$(\text{numVars (Var \_)}) + (\text{leftHeight (Var \_)}) \leq \text{size (Var \_)}$$

- **Left-hand side:**

$$\begin{aligned}(\text{numVars (Var \_)}) + (\text{leftHeight (Var \_)}) &= 1 + (\text{leftHeight (Var \_)}) && \text{(by numVars.2)}\\ &= 1 + 0 && \text{(by leftHeight.2)}\\ &= 1 && \text{(by arith.)}\end{aligned}$$

- **Right-hand side:**

$$\begin{aligned}\text{size (Var \_)} &= 1 && \text{(by size.2)}\\ &\geq 1 && \text{(by arith.)}\\ &\geq \text{L.H.S} && \text{(by replacement)}\end{aligned}$$

Thus it shows that the left-hand-side is smaller than or equal to the right-hand-side, which completes the proof of the second base case.

Since all base cases are proved, we completed the proof of the base case of the statement. Now we prove the induction step:

- **Assume:**

$$\text{(numVars e1) + (leftHeight e1)} \leq \text{size e1} \qquad \text{(hyp.1)}$$
$$\text{(numVars e2) + (leftHeight e2)} \leq \text{size e2} \qquad \text{(hyp.2)}$$

- **Want:**

$$\text{(numVars (Add e1 e2)) + (leftHeight (Add e1 e2))} \leq \text{size (Add e1 e2)}$$

- **Left-hand side:**

$$\begin{aligned} L.H.S &= \text{(numVars (Add e1 e2)) + (leftHeight (Add e1 e2))} \\ &= \text{(numVars e1) + (numVars e2) + (leftHeight (Add e1 e2))} \\ &\qquad \text{(by numVars.3)} \\ &= \text{(numVars e1) + (numVars e2) + 1 + (leftHeight e1)} \quad \text{(by leftHeight.3)} \end{aligned}$$

- **Right-hand side:**

$$\begin{aligned} R.H.S &= \text{size (Add e1 e2)} \\ &= 1 + \text{(size e1) + (size e2)} && \text{(by size.3)} \\ &\geq 1 + \text{(numVars e1) + (leftHeight e1) + (size e2)} && \text{(by hyp.1)} \\ &\geq 1 + \text{(numVars e1) + (leftHeight e1) + (numVars e2) + (leftHeight e2)} \\ &\qquad \text{(by hyp.2)} \\ &\geq 1 + \text{(numVars e1) + (leftHeight e1) + (numVars e2)} && \text{(by arith.)} \\ &\geq L.H.S && \text{(by replacement)} \end{aligned}$$

  Therefore, we have proved that the left-hand-side is smaller than or equal to the right-hand-side, on the assumption that the induction hypothesis holds, which completes the induction step.

Since the induction step is completed, we can say the proof itself is also completed. □