Problem Set #4 CSC236 Fall 2018

Mengning Yang, Licheng Xu, Chenxu Liu

November 23, 2018

We declare that this assignment is solely our own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters.

This submission has been prepared using LaTeX.

Problem 1.

(Warmup - this problem will NOT be marked).

Here is code for a recursive function that finds the minimum element of a list.

```
def rec_min(A):
  if len(A) == 1:
    return A[0]
  else:
    m = len(A) // 2
    min1 = rec_min(A[0..m-1])
    min2 = rec_min(A[m..len(A)-1])
    return min(min1, min2)
```

State preconditions and postconditions for this function. Then, prove that this algorithm is correct according to your specifications.

Problem 2.

(10 MARKS) **Iterative Program Correctness.** One of your tasks in this assignment is to write a proof that a program works correctly, that is, you need to prove that for all possible inputs, assuming the precondition is satisfied, the postcondition is satisfied after a finite number of steps.

This exercise aims to prove the correctness of the following program:

```
def mult(m,n) : # Pre-condition: m,n are natural numbers
""" Multiply natural numbers m and n """
1.     x = m
2.     y = n
3.     z = 0
4.     # Main  loop
5.     while not x == 0 :
6.         if x % 3 == 1 :
7.             z = z + y
8.         elif x % 3 == 2 :
9.             z = z - y
10.            x = x + 1
11.        x = x div 3
12.        y = y * 3
13.    # post condition: z = mn
14.    return z
```

Let `k` denote the iteration number (starting at 0) of the `Main Loop` starting at line 5 ending at line 12. We will denote I_k the iteration k of the `Main Loop`. Also for each iteration, let x_k, y_k, z_k denote the values of the variables x, y, z at line 5 (the staring line) of I_k.

1. (5 Marks) **Termination**. Need to prove that for all natural numbers n, m, there exist an iteration k, such that $x_k = 0$ at the beginning of I_k, that is at line 5.

   HINT: You may find helpful to prove this helper statement first:

   For all natural numbers k, $x_k > x_{k+1} \geq 0$. (Hint: do not use induction).

2. (2 Marks) **Loop invariant**

   Let $P(k)$ be the predicate: At the end of I_k (line 12), $z_k = mn - x_k y_k$.

   Using induction, prove the following statement:

$$\forall k \in \mathbb{N}, P(k)$$

3. (3 Marks) **Correctness** Let $C(m, n)$ be the following predicate defined in the domain of natural numbers: Program `mult(m,n)` returns mn. Let $S(m, n)$ be the function equal to the number of steps of the program `mult(m, n)`. The statement that `mult(m,n )` is correct can be formulated in English as follows:

   The program `mult(m,n)` which takes as input any two natural numbers m, n computes the value mn after a finite number of steps.

   Write the symbolic statement that is equivalent to the English statement above and prove it (using (1) and (2)).

Answer:

1. In order to prove the termination of this loop, we need to prove two things:
   a). The variant is decreasing on every iteration of the loop.
   b). The variant is a natural number.
   Let x be the variant, x = m , m is a natural number by the pre-conditon, so condition b) satisfied.
   Then, let's see if x is decreasing in 3 seperate cases:
   Suppose X of k_{th} iteration is X_k, $X_k > 0$

   case 1:
   When $x_k \bmod 3 = 1$
   $X_{k+1} = X_k \text{ div } 3 = (X_k - 1) \text{ div } 3$
   Because $X_k > 0$, so $X_{k+1} = (X_k - 1) \text{ div } 3 < X_k$

   case 2:
   When $x_k \bmod 3 = 2$
   $X_k = X_k + 1$, $X_{k+1} = X_k \text{ div } 3 = (X_k + 1) \text{ div } 3 < X_k$

   case 3:
   When $x_k \bmod 3 = 0$
   $X_{k+1} = X_k \text{ div } 3 < X_k$, when $X_k = 1, 2$, $X_{k+1} = 0$, and the next iteration when $X_k = 0$ at the beginning of I_k, the loop terminates.

   So, $X_k > X_{k+1} > 0$, the variant is getting smaller on each iteration of the loop, and the variant is a natural number.
   Therefore, the loop will eventually terminate at some point.

2. $P(k)$: At the end of I_k (line 12), $Z_k = mn - X_k Y_k$.

Base case: $m = 0$, $n \geq 0$, where n is a nature number. We have x = m = 0, y = n, z = 0. So the predicate would be $z = mn - xy = 0$. Therefore, the base case holds.

Induction step:
Assume that at the end of one iteration of the loop we have $Z_k = mn - X_k Y_k$ by Induction Hypothesis. We want to prove that at the end of (k+1)th iteration $Z_{k+1} = mn - X_{k+1} Y_{k+1}$.

case 1: when $X\%3 = 1$

$$\begin{aligned}
z_{k+1} &= z_k + y_k \\
x_{k+1} &= x_k//3 = (x_k - 1)//3 \\
x_{k-1} &= 3x_{k+1} \\
y_{k+1} &= 3y_k \\
y_k &= y_{k+1}//3
\end{aligned}$$

plug in z_k and y_k

$$\begin{aligned}
z_{k+1} &= z_k + y_k \\
&= mn - x_k y_k + y_k \\
&= mn - (x_k - 1)(y_k) \\
&= mn - (3x_{k+1} + 1 - 1)(y_{k+1}//3) \\
&= mn - x_{k+1} y_{k+1}
\end{aligned}$$

case 1 is correct.

case 2: when $x\%3 = 2$

$$\begin{aligned}
x_{k+1} &= x_k + 1 \\
z_{k+1} &= z_k - y_k \\
x_{k+1} &= (x_k + 1)//3 \\
x_k &= 3x_{k+1} - 1 \\
y_{k+1} &= 3y_k \\
y_k &= y_{k+1}//3
\end{aligned}$$

plug in

$$\begin{aligned}
z_{k+1} &= z_k - y_k \\
&= mn - x_k * y_k - y_k \\
&= mn - (x_k + 1)y_k \\
&= mn - (3x_{k+1} - 1 + 1)y_{k+1}//3 \\
&= mn - x_{k+1}y_{k+1}
\end{aligned}$$

case 2 is correct.

case 3 : when $x\%3 = 0$

$$\begin{aligned}
z_{k+1} &= z_k \\
x_{k+1} &= x_K/3 \\
x_k &= 3x_{k+1} \\
y_{k+1} &= 3y_k \\
y_k &= y_{k+1}/3
\end{aligned}$$

plug in

$$\begin{aligned}
z_{k+1} &= z_k \\
&= mn - x_{k+1}y_{k+1} \\
&= mn - (y_{k+1}//3)(3x_{k+1}) \\
&= mn - x_{k+1}y_{k+1}
\end{aligned}$$

case 3 is correct.
Therefore,

$$\forall k \in \mathbb{N}, P(k)$$

.

3. a)

$$\forall m, \forall n \in \mathbb{N}, C(m,n)$$

. $\exists c \in \mathbb{R}^+, \forall n_0 \in \mathbb{N}, \text{such that } \forall n \geq n_0, C(m,n) \leq cS(m,n)$
where $|cS(m,n)| < \infty$, $C(m,n)$ holds.

b) loop invariant and correctness are proved in part 2.
c) the program's termination is proved in part 1.
Therefore, the program is correct.

Problem 3.

(6 MARKS) A *palindrome* is a string that is equal to its reversal: examples are 'a', 'wow', and 'abcdedcba'. Consider the following algorithm.

```
def longestPalindrome(s):
  '''
  Pre: s is a non-empty string
  Post: returns the longest palindrome that is a substring of s.
  If there is more than one palindrome in s of maximum length,
  return <YOU FIGURE THIS OUT>.

  >>> longestPalindrome('ballaaa')
  'alla'
  >>> longestPalindrome('ballaaaa')
  <YOU FIGURE THIS OUT>
  '''

  if len(s) == 1:
    return s
  else:
    palindrome1 = longestPalindrome(s[1..len(s)-1])
    palindrome2 = firstPalindrome(s)

    if len(palindrome1) > len(palindrome2):
      return palindrome1
    else:
      return palindrome2
```

You have two tasks here, which should be accomplished together.

- As is often the case in real life, the client (Ilir) has failed to consider an edge case in the provided specification. By studying the given algorithm, you must complete the specification.
- Once again, write pre- and postconditions for the helper function `firstPalindrome`, and then prove that `longestPalindrome(s)` is correct, assuming that `firstPalindrome` is correct.

Note that you cannot prove that `longestPalindrome` is correct without completing its specification; but in order to complete its specification, you can *carefully trace* *through the code*, as you would when actually proving correctness (so the two tasks can be accomplished together).

1.
```
def longestPalindrome(s):
  '''
  Pre: s is a non-empty string
  Post: returns the longest palindrome that is a substring of s.
  If there is more than one palindrome in s of maximum length,
  return the longest palindrome whose first index is smallest.

  >>> longestPalindrome('ballaaa')
  'alla'
  >>> longestPalindrome('ballaaaa')
  'alla'
  '''

  if len(s) == 1:
    return s
  else:
    palindrome1 = longestPalindrome(s[1..len(s)-1])
    palindrome2 = firstPalindrome(s)

    if len(palindrome1) > len(palindrome2):
      return palindrome1
    else:
      return palindrome2
```

2.
```
def firstPalindrome(s):
    '''
    Pre: s is a non-empty string.
    Post:return the palindrome containing S[0].
    '''
    pass
```

3. path 1:
   As len(s)=1, the longest palindrome in s is s self.So return s is true.

   path 2:
   len(s)$\geq$ 2 $\implies$ s[1..len(s)-1] is not empty $\implies$ longestPalindrome(s[1..len(s)-1]) satisfy the precondition of longestPalindrome

s is not empty $\implies$ firstPalindrome(s) satisfy the precondition of firstPalindrome

len(s[1..len(s)-1]) = len(s)-1 $\implies$ the recursive call is on a smaller input.

suppose longestPalindrome() satisfy the postcondition for $s_{i-1} = s_i$[1:]. let's consider about longestPalindrome(s_i):

palindrome1 = longestPalindrome(s_{i-1})=the first longest palindrome of $s_i[1:]$
palindrome2 = firstPalindrome(s_i)=the first palindrome of s_i

if len(palindrome1) $>$ len(palindrome2), then palindrome1 is still the first longest palindrome of s_i, $\implies$ return palindrome1 is correct.

if len(palindrome1) = len(palindrome2), then there is more than one palindrome in s_i of maximum length and palindrome2 is the one appears first $\implies$ return palindrome2 is correct.

if len(palindrome1) $<$ len(palindrome2), then palindrome2 is the only longest Palindrome $\implies$ return palindrome2 is correct.

Therefore, longestPalindrome() is correct.