Problem Set #4 CSC236 Fall 2018

Si Tong Liu, shuo Yang, Jing Huang

22/11/2018

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.

Solution:

1. Precondition: A is a list and len(A) > 0

2. Postcondition: return the minimum element of list A 3. Prove the correctness of the algorithm:

path 1: return A[0]

To enter this path, len(A) == 1, which means there is only one element in listA

so the minimum element is the only element in list A, holds

path 2: return min(min1, min2) To enter this path, len(A) ≥ 2

(1) then len(A) // 2 > 1 , precondition holds

(2) m = len(a) // 2, m is getting smaller, so smaller value holds

(3) for min1, assume list1 = A[0..m-1], so len(list1) = m-1-(-1) = m

for min2, assume list2 = A[m..len(A) - 1]m so len(list2) = len(A)-1 - (m-1) = len(A)-m

so that len(list1) + list(list2) = len(A)-m+m = len(A)

which means the minimum of min1 and min2 is indeed the minimum element of listA

Then we use complete induction to prove the recursive call

We assume the the recursive call holds for listA with length of n prove for n+1

[1] n is odd list1 becomes A[0..m],so len(list1) = m-(-1) = m + 1

list2 becomes A[m+1,n+1-1],so len(list2) = n - m

so that len(list1) + len(list2) = m + 1 + n -m = n + 1, which is the length of listA which means the minimum of min1 and min2 is the minimum element of listA, holds

[2] n is even list1 remains the same A[0..m], so len(list1) = m-1-(-1) = m

list2 becomes A[m..n+1-1], so len(list2) = n-(m-1) = n-m+1

so that len(list1) + len(list2) = n-m+1+m = n+1, which is the length of listA

which means the minimum of min1 and min2 is the minimum element of listA, holds

Therefore, the algorithm is correct

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)).

Solution:

1. According to the precondition, m,n are natural numbers and x = m, which means $x \geq 0$

Firstly, prove for all natural numbers k, $x_k > x_{k+1} \geq 0$

Case 1: when x = 0, the while loop does not even run. So for any pairs of numbers (n,0), there do exist x = 0 at I_0 with x = m = 0

Case 2: at iteration k, $x_k > 0$. After one more step of while loop, $x_{k+1} = x_k/3$.

(1) If $x_k < 3$, then $x_k//3 = 0 => x_{k+1} = 0$.

(2) If $x_k \geq 3$, then $x_k//3 > 0 => x_{k+1} > 0$

So that in this case, $x_{k+1} \geq 0$

Moreover, $x_k/3 < x_k$, which means as k is getting bigger, x is getting smaller.

Therefore, $x_k > x_{k+1} \geq 0$, holds. Then it is clear that $x_k > x_{k+1} \geq 0$ for all natural numbers k is true.

In conclusion, for any for all natural numbers n,m, there exists an iteration k, such that $x_k = 0$ at the beginning of I_k

2. We use complete induction to prove

(1) Base Case: When m = 0, $z_k = 0 - 0 = 0$, holds

(2) Predicate is P(k): At the end of I_k (line 12), $z_k = mn - x_k y_k$, $\forall k \in \mathbb{N}$

(3) Induction hypothesis: assume P(k) is true for all natural number k.

(4) Induction steps: Prove for k + 1:

if there exits $x_k + 1$, then x_k can not be 0

Case 1: $0 < x_k < 3$ so that at iteration I_{k+1}, $x_{k+1} = x_k//3 = 0$

so that z does not change,which implies to

$z_{k+1} = mn = mn - 0 = mn - x_{k+1}y_{k+1}$, holds

Case 2: $x_k = 3, 6, 9...$, where x_k mod 3 = 0 and we assume $x_k = 3n$,where n is any nature number.

In this case,$z_{k+1} = z_k = mn - x_k y_k$

$= mn - 3ny_k$ (we assumed $x_k = 3n$)

$= mn - n(3y_k)$

$= mn - x_{k+1}y_{k+1}$ (since $x_{k+1} = x_k//3 = 3n//3 = n$ and $y_{k+1} = 3y_k$)

Case 3: $x_k > 3$ and at iteration I_{k+1}, x_k mod 3 = 1 and we assume $x_k = 3n + 1$, where n is any nature number .

So that $z_{k+1} = z_k + y_k$ (by execute the I_{k+1} step in while loop)

$= mn - x_k y_k + y_k$ (by IH)

$= mn - (x_k - 1)y_k$

$= mn - (3n + 1 - 1)y_k$ (since we assumed $x_k = 3n + 1$)

$= mn - 3ny_k$

$= mn - n(3y_k)$

$= mn - x_{k+1}y_{k+1}$ (since $x_{k+1} = x_k//3 = (3n + 1)//3 = n$,line 11 and at I_{k+1}, $y_{k+1} = 3 * y_k$, line 12), holds

Case 4: $x_k > 3$ and at iteration I_{k+1}, x_k mod 3 = 2 and we assume $x_k = 3n + 2$, where n is any nature number.

So that $z_{k+1} = z_k - y_k$ (by execute the I_{k+1} step in while loop)

$= mn - x_k y_k - y_k$ (by IH)

$= mn - (x_k + 1)y_k$

$= mn - (3n + 3)y_k$ (since we assumed $x_k = 3n + 2$)

$= mn - (n + 1)(3y_k)$

$= mn - x_{k+1}y_{k+1}$ (since $x_{k+1} = (x_k + 1)//3 = (3n + 3)//3 = n + 1$, line 10 and 11. Also $y_{k+1} = 3y_k$, line 12), holds

Therefore, it is true that at the end of I_k (line 12), $z_k = mn - x_k y_k$

3. According to 2 above, $z_k = mn - x_k y_k$.

According to 1 above, x_k will decrease as function goes on until $x_k = 0$, which meas the steps are finite.

So that when $x_k = 0$, the while loop terminates, and we get $z_k = mn - 0 = mn$, which is the same of postcondition, holds.

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

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).

Solution:

1. for `firstPalindrome`

precondition: the input is a non-empty string

postcondition: return the first palindrome of the input string from the most left

2. for `longestPalindrome`

Postcondition: returns the longest palindrome that is a substring of s. If there is more than one palindrome in s of maximum length, return the rightmost palindrome.

To prove the correctness of `longestPalindrome`:

[1] Path 1: return s

When s is a character, the palindrome of s is itself. So it holds for returning s.

[2] Path 2: return palindrome1

(1) to enter path2 , length of s is greater than 1. So precondition holds

(2) s[1..len(s)-1], the input string starts from the second character to the end, the value is getting smaller, so smaller value holds

(3) Then we use complete induction to prove:

Firstly, Predicate P(k): `longestPalindrome` returns longest palindorome of s, where the length of input s is k, $k \geq 1$.

Secondly, Induction hypothesis: for $k \geq 1$, `longestPalindrome` returns longest palindorome of s

Thirdly, Induction steps: prove for P(k+1)

By IH, when s has lenght of k, the longest palindrome is palindrome1. When we add one more string, we compare the palindrome1 with the left most palindrome2 that is generated by `firstPalindrome`, since in this path, length of palindrome1 is greater than palindrome2, holds

[3] path 3: return palindrome2

proof steps are similar to path2, however, to enter path 3, the length of palindrome1 is less than or equal palindrome2. holds

Finally, check postcondition, if there are no palindromes with same maximum length, we will keep comparing palindrome1 and palindrome2 until the end of input string. But if there are palindromes with same maximum length, we will enter path 3 and get the palindrome whose index is more right than the previous one. holds