Programmering for computerteknologi Hand-in Assignment Exercises

Week 11: Verifying Correctness of Recursive Programs

Written by: Alexander A. Christensen (202205452)

Disclaimer: Due to errors with CMake that neither me, nor the TAs have solved, the test-cases have not been run. Instead, the functions have been manually tested.

The code can still be found at https://github.com/Aarhus-University-ECE/assignment-11-A-CHRI

Exercise 1

We wish to prove by induction that the following code computes the factorial of n.

```
/* Factorial function definition */
int fact(int n)
{
    /* pre-condition */
    assert (n >= 1);

    /* post-condition */
    if(n > 1)
        return n * fact(n - 1);
    else
        return 1;
}
```

We do this by first proving the base case `n = 1`, and then the recursive step `n > 1`.

Base case: If n is set to one the program will return 1. We can verify that $1! = 1$. Therefore the base case must hold.

Recursive step: In case of recursion the program returns `n * fact(n-1)`. For $n = 2$ this results in $fact(2) = 2 * fact(1) = 2 * 1 = 2$, which is equal to $2!$.

We now approach the recursive step with an unknown variable k, giving us `fact(k) = k * fact(k-1)`. We know that this statement is true for $k > 1$. For`fact(k)` to hold, `fact(k-1)` also needs to hold. For the time being we then assume that `fact(k-1)` is correct. In this case the function becomes `fact(k) = k * [product of integer 1 to k-1], for k > 1`.

We've now proven that:

- `fact(1)` is correct
- `fact(k)` is correct **IF** `fact(k-1)` is correct for $k > 1$

Using these statements, we can pick any integer e.g 3, which results in `fact(3)` being correct **if** `fact(2)` is correct **if** `fact(1)` is correct, which we know it is. Therefore all the previous statements must be correct.

We can pick any number for $k > 1$, and we will always end at `fact(1)`, verifying that `fact(k)` works for any positive integer k.

Exercise 2

We notice that the program computes the sum of the first n positive odd numbers equaling n^2. Using the inductive proof given, we recognize the base case, as well as the recursive step. using these we can write the recursive equation for the function.

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ (2 \cdot n - 1) + f(n-1), & n > 1 \end{cases}$$

Using this we can implement our code.

```
int sumn (int n)
{
    /* Pre: Positive integer */
    assert(n > 0);

    /* Base case: */
    if( n == 1 ) {
        return 1;
    }

    /* Recursive step: */
    else {
        return 2 * n - 1 + sumn(n - 1);
    }
}
```

Exercise 3

Given the following script to compute the sum of integers from 1 to n, we wish to implement a script, with the same function, using tail recursion, as well as a while loop.

```
/* Sum integers 1 to n */
int sum(int n)
{
    /* pre-condition */
    assert (n >= 1);

    /* post-condition */
    if(n > 1)
        return n + sum(n - 1);
    else
        return 1;
}
```

Tail recursion

```
/* Sum integers 1 to n */
int sumtail(int n, int total)
{
  /* pre-condition */
  assert(n >= 1);
  /* post-condition */
  if (n > 1)
    return sumtail(n - 1, total + n);
  else
    return 1 + total;
}
```

While-loop

```c
/* Sum integers 1 to n */
int sumwhile(int n)
{
  /* Pre: positive integer */
  assert(n > 0);

  /* Counter variables */
  int count = 1;
  int total = 0;

  /* While loop */
  while(count <= n) {
    total += count;
    count++;
  }

  return total;
}
```

Testing

To test the functionality of the scripts we're testing the following three scenarios

- Sum of 1 — Should be equal to 1
- Sum of 3 — Should be equal to 6
- Sum of 10 — Should be equal to 55

Exercise 4

We wish to implement a function for finding the n'th number in the Fibonacci sequence using tail recursion. We do this by implementing two counter into the function itself, like shown in the implementation below.

```c
/* Fibonacci function definition */
int fib (int n, int p, int pp)
{
    /* Pre: Positive integer */
    assert(n > 0);

    /* Cases where n < 3 */
    if (n == 1)
        return p;
    if (n == 2)
        return pp;

    /* Recursive step */
    else {
        return fib(n - 1, pp, p + pp);
    }
}
```

Testing

To test the functionality of the script we're testing the following three scenarios

- Number 1 in the sequence — Should be equal to 0
- Number 2 in the sequence — Should be equal to 1
- Number 6 in the sequence — Should be equal to 5