School of Innovation, Design and Engineering
27 March 2018
DVA422 - Software Engineering 3

Albert Bergström
abm13007@student.mdh.se

Kevin Oswaldo Cabrera Navarro
kco17001@student.mdh.se

Laboration 4: Formal Specification

Table of Contents

1 Formal Specification 2

2 References 3

Laboratory 4

(estimated time 4:00).

Based on the Chapter 27 of *Software Engineering*, by Ian Sommerville the informal description of a *List* system operations is:

- **Create**: Brings an empty list into existence
- **Cons**: Creates a new list with an added member
- **Length** Evaluates the list size
- **Head:** Evaluates the front element of the list.
- **Tail**: Creates a list by removing the head from its input list [1].

For Laboration 4.1 we were assigned to extend the type *List* with two extra operations, both will work only with non-empty lists. The first operation is **CircShift.** It removes the first element and appends it into the end of the list. This operation takes a number *n* as a parameter and return the result of the circular shift *n* times. When the **CircShift** has a parameter of zero or the same length of the given list, there is no shifting since it will return the same result in the end. This is an example of the execution and its signature:

- CircShift(myList, 0) = [1, 2, 3]
- CircShift(myList, 1) = [2, 3, 1]
- CircShift(myList, 3) = CircShift(myList, 0) = [x, y, z]
-
- CircShift(List,Integer) -> List (This is the signature for the Circular Shift operation)

When we translated the operation into F# code we have the next **CircShift** recursive function:

```
let rec CircShift (l, n) =
   if n%Length l = 0 then
      l
   else
       CircShift(Cons(Tail(l),Head(l)),n-1)
```

The axiom CircShift(L,n) = **if** length(L) % n=0 **then** L **else** CircShift(Cons(Tail(L), Head(L)), n-1) is performing the circular shifting operation by taking *L* as the list argument and *n* as the parameter for the number of times a shifting is required. Afterwards, it does a conditional statement of whether *n* is the same as the modulus of the lists length. If that is the case then the list is returned without any change, if it is not, then the circular shift is applied. To have a recursive function without an infinite loop risk, we subtract 1 to *n* every time the **CircShift** is called recursively .

The second operation is **Count**, given a list and an element it returns the number of times the element appears in the list, this in an example of the execution:

- Count([1,2,2,3,1], 1) = 2
- Count([1,2,2,3,1], 3) = 1
- Count([1,2,2,3,1], 4) = 0

To make it count the values properly, we added the function **_Count** which is called by **Count**. The signatures of the operations are:

- ➢ Count(List,Element) -> Integer
- ➢ _Count(List,Element,Integer) -> Integer

Count gives the integer parameter in **_Count** a starting value of zero. Then, **_Count** goes through the list, increasing its integer whenever it matches a value in the list with the parameter. When the list is empty, **_Count** returns its value which is in then returned by **Count**.

The axioms of the operations **Count** and **_Count** are:

- ➢ Count(L,v) = _Count(L,v,0)
- ➢ _Count(Create, v2, i) = i
- ➢ _Count(Cons (L, v1),v2,i) = **if** v1=v2 **then** _Count(L,v2,i+1) **else** _Count(L,v2,i)

When we translated the operation into F# code we have the **Count** function and the **_Count** recursive function:

```
let rec _Count (l,n,i) =
   match l with
   |   Create -> i
   |   Cons(l,v) -> if v=n then _Count(l,n,i+1) else _Count(l,n,i)
let Count (l, n) =
   _Count(l,n,0)
```

References

[1] *Ian Sommerville, "Formal Specification," Chapter 27 of Software Engineering, 9th edition, Addison-Wesley, 2010.*