Advanced Programming IT712A
Assignment 01

Sergey Redyuk

University of Skövde
Sweden
b16serre@student.his.se

1 Exercise

Consider the definition of a list of all possible combinations of digits $\{0, 1\}$ *with length* $n \geq 1$. *That is, given* $n = 2$ *we need a function to return the following elements:*

$List[List[Int]] = List(List(0,0), List(0,1), List(1,0), List(1,1))$

and for $n = 3$ *we need a function to return:*

$List(List(0,0,0), List(0,0,1), List(0,1,0), List(0,1,1), List(1,0,0), List(1,0,1), List(1,1,0), List(1,1,1))$

Consider a more general definition where all possible combinations are computed using digits from a list. For example, the following call $combinationList(2, List(0,1,2))$ *will return*

$List(List(0,0), List(0,1), List(0,2), List(1,0), List(1,1), List(1,2), List(2,0), List(2,1), List(2,2))$

More specifically,
– *Implement a recursive function with the name combination that return this list. We should be able to call it as follows:*

$combination(2)$
$combination(3)$

– *Define the most general case where all possible combinations from a list of digits is considered.*
– *Show examples of their use. We should be able to call it as follows:*

$combinationList(2, List(0,1,2))$

Solution

The first solution for *combination* function is based on the idea of "expanding" the list given as follows:

$List(0) \Rightarrow List(List(0,0), List(0,1))$
$List(0,1,0) \Rightarrow List(List(0,1,0,0), List(0,1,0,1))$

The recursive strategy is used to generate all combinations of given length. Each recurrent call increments the size of arrays of type $List[Int]$. Then the recursive approach (*go* function) is used to traverse the base case - $List(List(0), List(1))$ of type $List[List[Int]]$.
The recursion stops when depth of recursion equals to *number* — the parameter given.

The output is the array of type $List[List[Int]]$ which represents a concatenation of sub-arrays returned by each nested recurrent call and includes all possible combinations of $\{0,1\}$ with the length of *number*.

The second solution (commented code) is based on the *baseChange* function which converts a number into binary form. To retrieve all combinations of $0,1$ using this function next approach should be used:
convert each number from range 0 *to* $2^{number}-1$ (2^{number} represents the amount of all combinations of $\{0,1\}$) into binary form and store all results in one array of type $List[List[Int]]$.

The *combinationList* function is a general form of the *combination* function with modified *expand* part — the array given expands as follows:

$List(0) \Rightarrow List(List(0,0), List(0,1), List(0,2), List(0,3))$ for the parameter *list* equals to $List(0,1,2,3)$
$List(0,1,0) \Rightarrow List(List(0,1,0,0), List(0,1,0,1), List(0,1,0,2))$ for the parameter *list* equals to $List(0,1,2)$

The $listify$ function is used to convert the array given of type $List[int]$ to array of type $List[List[Int]]$ in order to match the form of input required for correct work of the algorithm discribed for the function *combination*.
The cases such as *number* given is negative or array given is empty are considered.
Solution is based on both lectures notes and the "Scala Lecture Notes" handout — *recursion, pattern matching, lists, lazy and eager evaluation*.

2 Exercise

Consider a function that given a number and a base makes a base conversion and writes the number into the new base. Given the number n and the base b, the function should return a list of numbers between 0 *and* $b-1$ *corresponding to the representation in the new basis. For example, consider the following calls to the function and the corresponding results (related to change* 4532 *and* 45 *to base* 10 *and base* 2*, respectively).*

```
scala> baseChange (4532, 10)
res34: List[Int] = List(4, 5, 3, 2)

scala> baseChange (45, 2)
res35: List[Int] = List(1, 0, 1, 1, 0, 1)
```

As a summary,
– *Implement the function baseChange recursively.*

Solution

Solution for *changeBase* function is based on the algorithm of conversion between bases of a number — divide the number by the value of base recursively until number equals to 0, collect remainders of the division and store them in array (*inverted order*). For instance,

$45 = 22 \times 2 + 1 = (11 \times 2 + 0) \times 2 + 1 = ((5 \times 2 + 1) \times 2 + 0) \times 2 + 1 = (((2 \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1 = ((((1 \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1 = (((((0 \times 2 + 1) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1 \rightarrow 101101$

The recursive strategy (*go* function) is used with the base case *number* $= 0$ and the step

of recursion *number/base*.

The cases such as *number* given is negative or *base* given is negative are considered.
Solution is based on both lectures notes and the "Scala Lecture Notes" handout — *recursion, pattern matching, lists, lazy and eager evaluation*.