Declarative Programming and (Co)Induction
Haskell exercises

Davide Ancona and Elena Zucca

PhD Course, DIBRIS, Univ. Genova, June 23-27, 2014

Preliminaries The interactive interpreter is called *ghci*; under Windows, we suggest to use *WinGHCi*, which is "Windowsish". Running either one, you should get a prompt where you can write Haskell code. For instance, try:

```
3*2
```

or

```
(\x -> x+1) 41
```

With :? you can get the help of available (ghci) command, anyway the only ones you probably need are:

- :l, to load a file; for instance:

  ```
  :l c:\users\elena\Desktop\foo.hs
  ```

  You can also double-click on a `.hs` file, to start WinGHCi and load the file, or choose "Load..." from the File menu inside WinGHCi
- :r, to re-load the current file
- :t, to see the type of an expression
- :set +t, to enable the printing of types, after evaluation (note: the special variable it keeps the value of the last successful evaluation)

Exercises for beginners Define the following functions (we suggest to collect your definitions in a file) and then evaluate the given expressions, checking their results.

1. the identity function myid
   - myid 1
   - myid True
2. the function prod, which multiplies two integers
   - prod 3 4
3. the function twice that doubles an integer
   - twice 3
4. the predicate isEven which holds when an integer is even
   - isEven 3
5. the composition of two functions compose
   - compose isEven id 2
   - compose isEven id 3
   - compose isEven id
     - Note: you can't "print" this one
6. copy n e = the list consisting of n copies of e
   - copy 5 "ciao"

   (in Haskell strings are just lists of characters)

7. mysum g n = the sum for i from 0 to n of $g(i)$
8. sumsquare = the sum for i from 0 to n of $i * i$ as an *instance* (that is, obtained by partial application) of the previous function
9. forloop n body s = execute n times body starting from s
   - forloop 2 (\x -> x+1) 5
     (result: 7)
10. copy as an instance of forloop
11. the function leq which, given two functions f and g from integers to integers, checks whether f<=g for integers from n to m
    - leq id twice 1 10
12. prodlist = the product of a list of integers
13. after having defined the function itlist (as we have seen during the lecture), prodlist as an instance of itlist
14. member e l = checks whether e is a member of the list l
15. member as an instance of itlist
    - member 2 [1, 2, 3]

More challenging exercises

1. mydrop n l = removes the first n elements of the list l
2. myfilter p l = the list of the elements of l where the predicate p holds
3. poslist l = positive elements of the list (as an instance of the previous function)
4. forall p l = checks whether the predicate p holds for all elements of list l
5. allpos l = checks whether all elements of a list of numbers are positive (as instance of forall)
6. split p l = produces two lists, the former consisting of the elements of l satisfying p, the latter consisting of the elements that do not satisfy the predicate
7. after having defined the function mymap (as we have seen during the lecture), a function that given a list of pairs, checks whether for all pairs the first element is equal to the second
8. upto (n, m) = the list of integers from n to m
9. flatten [l_1, ..., l_n] = l_1 ++ ... ++ l_n
10. flatten as an instance of itlist
11. exists p l = checks whether there is at least an element of list l satisfying p
12. listit = an iterator analogous to itlist we have seen during the lecture, but which starts iteration from the *last* element
13. composelist which returns the composition of a list of functions, as an instance of listit
14. combine ([x_1, ..., x_n], [y_1, ..., y_n]) = [(x_1,y_1), ..., (x_n,y_n)]
15. sublists l = the list of all sublists of l

    for instance: [1, 2, 3] has, as sublists, [] [1] [2] [1, 3] etc

    (variant: the list of all sublists consisting of adjoining elements)
16. prefixes l = the list of prefixes of l

    for instance: [1, 2, 3] has, as prefixes, the lists [] [1] [1, 2] [1, 2, 3]