Functional Programming 1 — 1DL330

Assignment 2

Lab: Thursday, 19 September

Submission Deadline: 18:00, Wednesday, 25 September, 2019

This assignment should be solved in **groups of two** students. Please join a group in the *Assignment 2* group division on the Student Portal. If you have not found a group partner by the start of the lab for this assignment, please inform the lab assistant. If you would prefer to solve this assignment individually due to special circumstances, please contact the course teacher, Tjark Weber.

Instructions

- Start by reading the General Lab Instructions (on the Student Portal).
- Download the file `Lab2.hs` from the Student Portal. Modify this file as needed to add your solutions. Do *not* rename the file.
- The file that you submit must be a valid Haskell file, so the answers to some of the questions need to be given as comments using `--...` or `{- ... -}`.
- Remember to follow our Coding Convention (also on the Student Portal), and provide a specification for every function that you write. Also provide a variant for every recursive function.
- Make sure that your solutions pass the tests in `Lab2Tests.hs` before you submit. A submission that does not pass these tests will get grade K. See the *Testing* section near the end of this assignment for details.

1 Polymorphism

What is the type of the following expressions? (Try to remember/figure out the type yourself. Use GHCi's `:t` only to check your answer, or if you are stuck.)

1. `head`
2. `tail`
3. `\x -> x`
4. `(,)`
5. `(:)`

6. [[]]
7. **tail** [[]]
8. **id** : []
9. **id id**
10. **head** [**id**] "foo"

Which of these expressions are polymorphic?

2 Iota

Define a function iota so that iota n returns the list [0, 1, 2, 3, ..., n−1]. For example, iota 5 should return [0, 1, 2, 3, 4].

3 Intersection

Finite sets of integers can be represented by lists of integers. For this exercise we only consider lists without duplicates, i.e., every element is contained in the list at most once.

Sometimes it may be useful to also require that the lists are ordered.

- Examples of ordered lists: [6, 17, 23, 89], [], [−3, −2, −1]
- Example of a list that is not ordered: [17, 23, 6, 89]

1. First, do not assume that the input lists are ordered. Define a function inter s1 s2 that returns a list representing the set $s1 \cap s2$, i.e., the set containing the elements that occur in both argument lists. This is also known as the intersection of the two sets.

   Hint: You can use the function **elem** (from the Haskell Prelude) to test whether a given value occurs in a list.

2. Now, assume that the input lists are ordered. Naturally, to compute the intersection of two ordered lists, the previous solution will still work. However, it is possible to write a more efficient function interOrdered s1 s2 that assumes that the lists are ordered, and only visits each element of either list once (by recursing over both s1 and s2 simultaneously). Define interOrdered. (Note that the list returned by interOrdered should also be ordered.)

3. Verify that interOrdered is indeed faster than inter by testing on some long lists. Using the function iota it is easy to create lists with, say, 100 000 or 1 000 000 elements. (If your implementation of iota is well-written, it should be able to create lists this long quickly.) The file `Lab2.hs` contains the following declarations:

```
s1 = iota 100000
s2 = iota 1000000

t1 = inter s1 s2
t2 = interOrdered s1 s2
```

In a fresh GHCi session, enter `:set +s` to enable the printing of timing/memory stats after each evaluation. Then load `Lab2.hs`, evaluate

```
length s1
length s2
length t1
length t2
```

and report the stats printed by GHCi for each of these expressions.

4 Wildcard Matching[1]

For this exercise, we define a *string pattern* to be a string. String patterns may contain '?', '∗' and any other character. (Note that this definition of string patterns is *not* related to patterns for values of type **String** in the Haskell language.)

String patterns can *match* a string: '?' matches any single character, and '∗' matches any sequence of characters (including the empty sequence). Other characters in a string pattern only match themselves. The empty pattern only matches the empty string.

Implement a function isMatch :: **String** −> **String** −> **Bool** such that isMatch s p returns **True** if (and only if) the string pattern p matches the *entire* string s.

Examples:

- isMatch "aa" "a" == **False** because "a" does not match the entire string "aa"
- isMatch "aa" "∗" == **True** because '∗' matches any sequence of characters
- isMatch "cb" "?a" == **False** because '?' matches 'c' but 'a' does not match 'b'
- isMatch "adceb" "∗a∗b" == **True** because the first '∗' matches the empty sequence, and the second '∗' matches "dce"

Testing

Use the test cases provided in `Lab2Tests.hs` (from the Student Portal) to test your solution. See the instructions for Assignment 1 for further information on running the tests.

Some tests might fail because they assume other datatype definitions or function signatures than what you are using. If necessary, *change your solutions* to make all tests compile and run successfully.

Good luck!

[1] This exercise was inspired by Problem 44 at LeetCode, a web site that offers hundreds of challenging programming problems.