Programming Languages: Functional Programming Practicals 1. Program Synthesis

Shin-Cheng Mu

Spring 2022

1. Go to our course homepage `https://scmu.github.io/plfp/`. Navigate to the page Syllabus.
2. If you managed to install QuickCheck, download `practicals_01_code.zip`. Otherwise download `practicals_01_code_no_quickcheck.zip`. Uncompress the file, and change to the directory.
3. Let `n` be a digit between 0 and 9. Load the file `Tn.hs` by the command `ghci Tn.hs`.
4. This module imports a number of functions. Among them is a function `f0`. Find out its type, and try some inputs.
5. Your task is to define, in `Tn.hs`, a function that is identical to `f0`. Use your favourite editor to open the file `Tn.hs` and define your function there. (You have to give your function a different name, since `f0` is already in use.)
6. Hint:
   - Try exploiting the functions mentioned in the previous two lectures, including those in the work sheet. Compose them to form larger functions.
   - You do not need to define recursive functions at all. All the exercises can be completed by simply composing existing functions. I do not mind if you come up with some recursive definitions, however.
   - For now, forget about efficiency. There is no need to come up with the most efficient algorithm. This exercise is about building specifications using existing functions.
   - The solutions could be rather short — usually one or two lines per function.
   - Of course, you are not allowed to simply say `solution = f0` (or referring to functions defined in `Mn.hs`, copying their definitions, etc). Other than that, you can construct your target function in whatever ways you like.

- Some `f0` may have a type looking like `Eq a => T`. Ignore the part `Eq a` for now, and assume that it has type `T`. It may help you to know, however, that `Eq a` suggests that `f0` tests for equality somewhere...

7. **Auxiliary Functions**. It could be rather difficult seeing what `f0` is about. In all files there is an auxiliary function `f1`, which could be easier than `f0`. You may try constructing `f1` first. Typically, `f0` may then call `f1`.

8. In some files there is even an `f2`. Use `:t f2` to see whether it exists. If it exists, it may be useful for `f1` or `f0`.

9. The goal is to construct a function identical to `f0`. Functions `f1` (and `f2`, if exists) are merely there to give you hints.

10. **Testing**. The following applies only if you can install QuickCheck. How do you know whether your function is correct?

    - There is a function, `correct0`, defined in `Tn.hs`, specifying that it should produce the same result as `f0` for any input.
    - Assuming that your function is called `solution`, try `quickCheck (correct0 solution)` in `ghci`.
    - If the output says `+++ OK, passed 100 tests.` Congratulations!
    - Otherwise, QuickCheck gives you counterexamples when your function do not agree with `f0`.
    - There is also a `correct1`, to be used in the same way to check whether a given function is identical to `f1`. And a `correct2` as well.

11. If you find it too easy... you may download and try other exercises. Or try the challenge in `TChallenge.hs` and `MChallenge.hs`, where a function `find` is defined. As the name suggests, `find xs ys` finds the first occurrence of `xs` in `ys`, and returns the rest of the list. Can you define the same function? As before, you do not need to come up with a very efficient algorithm. It is preferable to specify the problem in the clearest way.