Haskell Tutorial: Tips and Tricks

October 1, 2019

0.1 Tips and Tricks

This is a somewhat unstructured collection of notes that may be useful. Consider this more a scratch space instead of a well thought out sequence of thoughts. If you thing more tips and tricks should be included, drop me a note and I will see whether I can add additional material.

0.1.1 Debugging

Most of the following is true for any programming language. So take the tricks and the advise serious and try to follow them.

Step #0 Compile your code. Code that has never seen a compiler in most cases does not work. There are even online Haskell interpreter that do not require any installation, simply google. There is no excuse for submitting code that has not seen a compiler.

Step #1 Turn on warnings (-Wall in ghci) and pay attention to all warnings and errors. Fix things. Iterate until the compiler is happy. And no, the compiler is not broken. The chances that a beginner finds a serious compiler bug is low.

Step #2 Write test cases (if not done yet). Automate your tests so you can re-run them easily after each and every change. Yes, automate, re-run all tests after each and every change. Ad-hoc testing is a waste of your time. So automate testing, stop wasting time.

Step #3 If your code does not pass your collection of test cases (your test suite) and it is not immediately clear what is wrong, you may need to go and figure what your code actually does:

1. If multiple functions are involved, test them separately. Go back to step 0 for each of them.
2. If you have a toy around, explain your code to the toy. (If no toy is available, use a student instead.) Explaining your code to someone (even to a toy) often reveals the bug. I am serious, try it, it often works.
3. If still no clue, it may help to obtain trace information. The `Debug.Trace.trace :: String -> a -> a` function may help. Consider this example:

[2]:
```
sum' :: [Integer] -> Integer
sum' [] = 0
sum' (x:xs) = x + sum' (tail xs)
```

```
sum' [1..10] == 55
```

```
False
```

Assuming you do not spot the error immediately, you can add the following import and another definition of sum′ to generate nice traces.

[4]:
```
import Debug.Trace

sum' :: [Integer] -> Integer
sum' xs | trace ("trace: sum' " ++ show xs) False = undefined
sum' [] = 0
sum' (x:xs) = x + sum' (tail xs)

sum' [1..10] == 55
```

```
False
```

This will show on the error output the following trace:

```
trace: sum' [1,2,3,4,5,6,7,8,9,10]
trace: sum' [3,4,5,6,7,8,9,10]
trace: sum' [5,6,7,8,9,10]
trace: sum' [7,8,9,10]
trace: sum' [9,10]
trace: sum' []
```

0.1.2 Testing