Testing: A Case Study in Testing a Stream Implementation Library

Read the entire task description before starting to work. There is no book chapter that we use this week, but you will need to refer to external documentation of testing libraries, and to lecture slides.

The task is to implement a test suite, for our implementation of lazy streams. In particular, test the following functions: `headOption`, `take`, `drop`, `map`, and `append`. Testing other functions will not be graded, although it might help you catch some bugs. Use a mixture of scenario tests and property tests as you see fit.

The file `StreamSpec.scala` contains an example of a such a mixed test suite. Examine this file before starting to work. Please also look how the homeworks in, for instance, 3rd week (Option) have been tested for inspiration (`ExerciseSpec.scala`).

You need to read up about the ScalaTest framework, used in this homework for unit testing and quick-check style property testing. Links to materials are in lecture slides.

We will test the effectiveness of your spec, by providing several broken and correct implementations for the above functions and measuring your success rate. We will manipulate your file by changing the imported implementation of Streams, so the only access to the implementation must be through the imports on top of the file (like in the example). Do not navigate to packages explicitly to access them.

It is a good idea to test on infinite streams, and probably a good idea to provide your own stream generator, but technically speaking it is possible without such a generator, just inconvenient. Testing that something is not forced is easiest done by injecting a side effect: make a stream that contains elements which when forced throw an exception. This is explicitly allowed in this homework.

It is fine if a test just lets an exception be thrown without interception (in the failing case). If an exception is thrown without being caught, we shall consider that the test has failed (so a bug has been detected).

Each exercise (almost) asks for a test. You can write more than one test in response, if you feel that one is not enough. Similarly it may happen that you have tested more than one property (solved more than one exercise) with a single test.

Hand-in: The `build.sbt` script is already set up the same way as in the automatic grading environment. Please do not change it. Also please refrain from making changes that would break the auto-grader (for instance do not move tests to another testing framework, rename the class, changing the top-level package, etc.) Prepare and hand in a single Scala file (`StreamSpec.scala`). No zip files are accepted.

Exercise 1. The test file contains an implementation of the following two properties for `headOption`: (i) it should return `None` on an empty stream, (ii) it should return `Some` for a non-empty stream. Familiarize yourself with their implementation.

The main source directory (`src/main/scala/fpinscala/laziness`) contains example implementations. The file `stream00/Stream.scala` is the book implementation of lazy streams, which we assume to be correct. (It must pass all tests in your hand-in, otherwise you loose 35% of points). The file `stream01/Stream.scala` contains an example that violates property 01. The file `stream02/Stream.scala` contains an example implementation that violates property 02. Both of these files should fail a test.

Try to run these tests on all three implementations and check when they fail. Understand why they fail. You can activate testing of any of these implementations by changing commented imports in the preamble of the spec file. (No coding required in this exercise)

Exercise 2. Test the following property for `headOption`: it should not force the tail of the stream. The place to add the test in the spec file is marked with a comment (`Exercise 2`).

Note that from now on we do not provide buggy implementations for your tests. This is like in reality: when writing tests you need to figure out yourself whether they test the property you wanted.

Exercise 3. Test that `take` should not force any heads nor any tails of the stream it manipulates.

Below `n` and `m` are assumed to be non-negative.

Exercise 4. Test that `take(n)` does not force the `(n+1)`st head ever (even if we force all elements of `take(n)`).

Exercise 5. Test that `s.take(n).take(n) ==s.take(n)` for any stream `s` and any `n`.

Exercise 6. Test that `s.drop(n).drop(m) ==s.drop(n+m)` for any `n`, `m`.

Exercise 7. Test that `s.drop(n)` does not force any of the dropped elements heads. This should hold even if we force some element in the tail.

Exercise 8. Test that `x.map(id) ==x` for any stream. Here `id` is the identity function.

Exercise 9. Test that `map` terminates on infinite streams.

Exercise 10. Test correctness of `append` on the properties you formulate yourself, understanding what the function should be doing.