Property-Based Testing

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 lazy lists. 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 `Exercise.scala` contains two example tests (See Exercise 1). Also all the previous weeks' tests (in `Exercises.test.scala`) follow the same style. Especially, the `Option` tests should be easy to read.

You need to read up about the ScalaCheck framework, used in this homework for unit testing and quick-check style property testing. **Warning!** Do not confuse it with similarly named ScalaTest framework (often an early hit on Google). ScalaTest is a different library and we are not using it. StackOverflow and GPT easily confuse ScalaCheck and ScalaTest.

The autograder will evaluate the effectiveness of your spec, by providing several broken (and correct) implementations for the above functions and measuring your success rate, how many of these you will be able to detect. Such broken implementations are often called "mutants" (for mutation testing). More mutants you kill with your tests, better your score. We will manipulate your file by changing the imported implementation of `LazyList`, 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 lazy lists, and probably a good idea to provide your own generator of lazy lists, 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 lazy list 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).

Most exercises ask to write 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: `Exercises.scala`

Documentation for the key types in Scalacheck:

- ScalaCheck: https://github.com/typelevel/scalacheck/blob/main/doc/UserGuide.md
- `Gen`, a key type: https://javadoc.io/doc/org.scalacheck/scalacheck_3/1.18.0/org/scalacheck/Gen.html
- `Arbitrary`, a key type: https://javadoc.io/doc/org.scalacheck/scalacheck_3/1.18.0/org/scalacheck/Arbitrary.html.
- I also find source code at https://github.com/typelevel/scalacheck/blob/v1.18.0/core/shared/src/main/scala/org/scalacheck/Gen.scala useful to read (actually I read it more often than the docs).

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

The `subjects/` directory contains three implementations of lazy lists. There, `lazyList00/LazyList`

`.scala` contains the book implementation that we assume is correct. It must pass all tests in your hand-in, otherwise you loose 35% of points. The file `lazyList01/LazyList.scala` contains an example that violates property `Ex01.01`. The file `lazyList02/LazyList.scala` contains an example implementation that violates property `Ex01.02`. Each of these implementations should fail a test.

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 or writing required in this exercise)

Exercise 2 [M]. Test the following property: `headOption` does not force the tail of a lazy list.

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 [M]. Test that `take` does not force any heads nor any tails of the lazy list it manipulates.

Exercise 4 [M]. In this and the in the following exercises, the numbers `n` and `m` are assumed to be non-negative. Test that `take(n)` does not force the `(n+1)`st head ever (even if we force all elements of `take(n)`).

Exercise 5 [M]. Test that `l.take(n).take(n) == l.take(n)` for any lazy list `s` and any `n`.

Exercise 6 [E]. Test that `l.drop(n).drop(m) == l.drop(n+m)` for any `n`, `m`.

Exercise 7 [M]. Test that `l.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 [M]. Test that `l.map(identity) == l` for any lazy list `l`. Here `identity` is the identity function.

Exercise 9 [M]. Test that `map` terminates on infinite lazy lists.

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