Monads for functional programming

This exercises are based on the first sections of *Monads for function programming* by Wadler.

Read the two first sections of *Monads for functional programming* by Phil Wadler (found in the repo). The paper's main point is that monads allow bringing some imperative programming style into functional programming, so functional programs are easy to modify in situations where imperative programs are easy to modify. It emphasizes how different side effects can be implemented in a referentially transparent manner, all in similar style. So we also learn how different programming language features can be modeled.

Wadler uses a langauge similar to Haskell to implement his evaluator. We will use Scala.

Hand-in: `Exercises.scala`

Exercise 1. Complete the implementation of `eval` with exceptions based on the paper. You will need a lot of pattern matching here. Pattern matching is common in implementation of language processing tools.

Exercise 2. Provide evidence that `M` (the implementation of the exception Monad in Wadler style) is a Monad in the sense of our course definitions. The test suite will automatically check whether it satisfies the monad laws. If you lack intuition, notice that `M` is essentially `Option` or `Either`, with `Return` being `Some`.

Exercise 3. Reimplement `eval` using `unit` and `flatMap` from the monad instance created in Exercise 2. This exercise will not work, if the previous is not implemented. You will still need pattern matching here, but on the language constructs only, not on the results.

Think: Why is `flatMap` suddendly available here, and it was not in Exercise 1?

Exercise 4. Implement `eval` once again but now using `for-yield`. We still use a bit of pattern matching on the language operators like above.

We cannot use the `for-if-yield` (mind the `if`) in the solution because we did not implement `withFilter` in `M`. This is easy to add, you can try, but this is not required material. see https://www.scala-lang.org/api/3.x/scala/collection/WithFilter.html#withFilter-fffffb75. The exercise can be completed without `withFilter`. It just takes 1–2 lines more.

Exercise 5. We are switching to Section 2.2 in Wadler's paper, implementing the evaluator with state. First, complete the implementation of the evaluator as per the spec in the paper. We do not have a monad instance for `M` yet, so we have to use pattern matching on the result. Remember that the State evaluators in the paper do not track divisions by zero (they track something different).

Exercise 6. Provide evidence that `M` is a `Monad` (the implementation of Wadler's state monad is a Monad as defined in our course). The test suite will automatically check whether it satisfies the monad laws. If you lack intuition notice, that `M` is essentially our `adpro.State`.

Note that we could implement the counter incrementation in `flatMap`, but this then would count not only divisions, so let's not do that.

Exercise 7. Now reimplement the `eval` of Exercise 5 using the instance of `Monad` (`flatMap` and `unit`) of Exercise 6.

Exercise 8. Reimplement the above using `for-yield`.

Exercise 9. We are moving to Section 2.4 in Wadler's paper, working on the output monad. Complete the implementation of the evaluator as per the spec in the paper. Again don't worry about divisions by zero or counting divisions, we only produce the trace in this exercise.

Exercise 10. Provide evidence that `M` is a `Monad` (the implementation of the output monad class is a `Monad`). The test suite will automatically check whether it satisfies the monad laws.

Exercise 11. Now reimplement the `eval` of Exercise 9 using `unit` and `flatMap`.

Exercise 12. Reimplement the above using `for-yield`.

Exercise 13. Read Sections 2.7-2.9 in the paper and revisit the monadic implementations (`evalMonad` and `evalForYield`) to make them in line with the paper. Can you get to the same point as Wadler, that changing a side effect requires a small change in the interpreter?

No new implementations in this exercise, just refactoring the 4 earlier exercises. But I encourage you to formulate a few thoughts (5-10 lines) on this experience. Was it valuable? Why?