CMPT 383: Vitamin #5

Anders Miltner
miltner@cs.sfu.ca

Due Oct 19

Introduction

This Vitamin is here to practice writing Functor, Applicative, and Monad instances, and to practice using Monad instances.

This submission will be autograded. There are some portions of the assignment that are ungraded, and some that will be graded. We provide a (partial) test suite for partial validation. You can run these tests by opening a terminal in the v4 directory, and running `stack test`.

We have not imported any functions. You are welcome to use or import anything in the Haskell standard library that may help. You do not need to import anything or use any built-in functions to finish this Vitamin.

1 `WarningAccumulatorMonad`

A WarningAccumulator contains two components, the data and the accumulated warnings. This data structure can work as a Functor, an Applicative, and a Monad.

When fmapping the values of a WarningAccumulator, the underlying data is updated by the provided function, and the warnings are left untouched.

The "pure" function of a WarningAccumulator as an Applicative should simply return a WarningAccumulator, with no warnings. The "apply" function of a WarningAccumulator should apply the function in the data of the left WarningAccumulator to the value in the data of the right WarningAccumulator. The warnings should be concatenated and returned – with the warnings on the left WarningAccumulator coming before the warnings on the right WarningAccumulator.

The "return" function of a WarningAccumulator is the same as the "pure" function. The "bind" function of a WarningAccumulator acts in 4 main steps. (1) The bind function extracts the warnings and the data from the provided input WarningAccumulator. (2) The bind function then applies the provided function to the data. (3) The bind function next extracts the warnings and the data from the newly returned WarningAccumulator. (4) The bind function finally returns a new WarningAccumulator with the data of the returned WarningAccumulator and the warnings of the input WarningAccumulator appended with the warnings of the generated WarningAccumulator.

2 `WarnedArithmetic`

Now we will be building an interpreter for a small language of floating point arithmetic. The terms in this language include Base values (which just contain floats), Division values (which divide the result of one expression by the result of another expression), and Plus values (which add the result of one expression by the result of another expression).

However, there's some dangers in Floating-point arithmetic. Floating point arithmetic will silently proceed when it performs operations that are considered invalid by other languages. Sometimes divisions by zero can create NaN values, sometimes they can create Infinity values, and sometimes they can create negative Infinity values. Sometimes performing additions with NaN values can create strange values. Because of this, we would like to create warnings whenever evaluating these expressions results in a division by zero, or an addition involving a NaN value.

For this section, you simply must complete the evaluate function. We have provided a stubbed-out version of the evaluate function that calls a helper, where the helper returns a WarningAccumulator. We have also defined other functions we think may be helpful to fill out: warningDivide and warningPlus.