Monad-Based Programming SS 2019

Assignment 4

Deadline for solutions: 27.06.2019, **12.15 a.m.**

Exercise 1 Triples* Need Not Be Kleisli (6 Points)

Complete the proof from the lecture that Kleisli triples bijectively correspond to the tripples (T, η, μ). To that end

1. define a Kleisli triple from a monad given as a triple (T, η, μ) and verify the axioms of Kleisli tripples;
2. show that the passage $(T, \eta, \_^*) \to (T, \eta, \mu) \to (T, \eta, \_^*)$ yields an identity;
3. show that the passage $(T, \eta, \mu) \to (T, \eta, \_^*) \to (T, \eta, \mu)$ yields an identity.

Exercise 2 Monads for Everyone (5 Points)

Choose one monad from the following list and prove that it is indeed a monad over the corresponding category. To that end use any of the equivalent definitions of a monad at pleasure.

1. *List monad (over Sets):* TX is the set of finite lists over X.
2. *State monad (over Sets):* $TX = (X \times S)^S$ for every set S.
3. *Exception monad (over any category with coproducts):* $TX = X + E$.
4. *Closure operator (over any poset-category $\mathcal{C}$, i.e. $Ob(\mathcal{C})$ is some partially ordered set and $Hom_{\mathcal{C}}(X, Y) = \{*\}$ if $X \leq Y$ and $Hom_{\mathcal{C}}(X, Y) = \{\}$ otherwise):* T satisfies properties:

$$\begin{array}{rcll} X \leq TX & & & \text{(extensiveness)} \\ X \leq Y & \text{implies} & TX \leq TY & \text{(monotonicity)} \\ TTX = TX & & & \text{(idempotence)} \end{array}$$

Exercise 3 Stateful Calculator (9 Points)

1. Write a Haskell function for parsing arithmetic expressions given by the following grammar:

$$e_1, e_2 = n \mid -e_1 \mid (e_1) \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \,/\, e_2$$

   where n ranges over positive natural numbers represented as nonempty strings of decimal digits. **Hint:** Reuse the example code from the lecture: https://www8.cs.fau.de/_media/ss15:mbp:parsing.hs.

2. Write a Haskell function evaluating the parse trees obtained in the previous clause to the corresponding integer values or to an error message if division by zero occurs. In any case record the evaluation history in the following style:

* "*Triple*" is an old term for "*monad*".

```
$ eval (calc "12 + 5 * 6")
$ Result 42, ["5 * 6 -> 30", "12 + 30 -> 42"]

$ eval (calc "12 + 5 / (0*6)")
$ Error, ["0 * 6 -> 0", "5 / 0 -> error"]
```

To this end, use an instance of the *exception monad transformer* from `Control.Monad.Except`

```
newtype ExceptT e (m :: * -> *) a
```

with an exception monad as the argument `m`.

Bonus Exercise (3 Points)

This only applies if you did not chose (3) in Exercise 2.
Show how to obtain (3) of Exercise 2 by modifying the solution of Exercise 1 of Assignment 3.