INFOAFP Assignments

AFP Assignment 2

Deadline: Friday, Nov 30, 2012, 24.00

General remarks

- Mail your solution to `doaitse@swierstra.net`, with in the subject "AsAFP-2010-2-4: *name1* and *name2*".
- Team size: preferably 2, but 1 is possible.
- For programs: Programs that are not type correct may not be graded. Programming style influences the grade.
- For text: Submit plain text or PDF, *not* HTML or Word.
- Gathering information on the internet is okay, but copying entire solutions from the internet (or elsewhere) is not allowed.
- You can make a Cabal package again, or just submit a zip file.

1 (35%)**.** Find Haskell definitions for the functions *start*, *stop*, *store*, *add* and *mul* such that you can embed a stack-based language into Haskell:

$$
\begin{aligned}
&p_1, p_2, p_3 :: \mathit{Int} \\
&p_1 = \mathit{start}\ \mathit{store}\ 3\ \mathit{store}\ 5\ \mathit{add}\ \mathit{stop} \\
&p_2 = \mathit{start}\ \mathit{store}\ 3\ \mathit{store}\ 6\ \mathit{store}\ 2\ \mathit{mul}\ \mathit{add}\ \mathit{stop} \\
&p_3 = \mathit{start}\ \mathit{store}\ 2\ \mathit{add}\ \mathit{stop}
\end{aligned}
$$

Here, p_1 should evaluate to 8 and p_2 should evaluate to 15. The program p_3 is allowed to fail at runtime.

Once you have that, try to find a solution that rejects programs that require non-existing stack elements during type checking.

Hint: Type classes are *not* required to solve this assignment. This is somewhat related to continuations. Try to first think about the types that the operations should have, then about the implementation.

2 (35%)**.** Consider the following class

$$\begin{aligned}&\textbf{class } \mathit{Splittable}\ a \textbf{ where}\\&\quad \mathit{split} :: a \to (a, a)\end{aligned}$$

for types that allow values to be split. Random number generators (for instance *StdGen*) allow such a split operation:

$$\begin{aligned}&\textbf{instance } \mathit{Splittable}\ \mathit{StdGen} \textbf{ where}\\&\quad \mathit{split} = \mathit{System.Random.split}\end{aligned}$$

We can also make other types an instance of *Splittable*. Define an instance $\mathit{Splittable}\ [a]$ where, assuming that the list passed is infinite, the list is split into one list containing all the odd-indexed elements, and one containing all the even-indexed elements of the original list.

Define an instance *Splittable Int* where n is split into $2 * n$ and $2 * n + 1$.

Consider the datatype

$$\textbf{data } \mathit{SplitReader}\ r\ a = \mathit{SplitReader}\ \{\mathit{runSplitReader} :: r \to a\}$$

which is isomorphic to the *Reader* datatype. Define a variant of the *Reader* monad

$$\textbf{instance } (\mathit{Splittable}\ r) \Rightarrow \mathit{Monad}\ (\mathit{SplitReader}\ r)$$

where the passed state is split before it is passed on. Also implement the instance of *MonadReader*:

$$\textbf{instance } (\mathit{Splittable}\ r) \Rightarrow \mathit{MonadReader}\ r\ (\mathit{SplitReader}\ r)$$

You have to pass enable the `FlexibleInstances` and `MultiParamTypeClasses` language extensions to make GHC accept this instance. The methods of the class *MonadReader* are

$$\mathit{ask} :: (\mathit{MonadReader}\ r\ m) \Rightarrow m\ r$$

that allows you to access the read state, and

$$\mathit{local} :: (\mathit{MonadReader}\ r\ m) \Rightarrow (r \to r) \to m\ a \to m\ a$$

that allows you to locally modify the read state.

Finally, consider the function

$$\begin{aligned}&\mathit{labelTree} :: \mathit{Int} \to \mathit{SplitReader}\ \mathit{Int}\ (\mathit{Tree}\ \mathit{Int})\\&\mathit{labelTree}\ 0 = \mathit{return}\ \mathit{Leaf}\\&\mathit{labelTree}\ n = \mathit{return}\ () \gg \mathit{liftM3}\ \mathit{Node}\ (\mathit{labelTree}\ (n-1))\ \mathit{ask}\ (\mathit{labelTree}\ (n-1))\end{aligned}$$

where

```
data Tree a = Leaf
            | Node (Tree a) a (Tree a)
  deriving Show
```

When calling *runSplitReader* (*labelTree* 3) 1, the function returns

```
Node (Node (Node Leaf 214 Leaf) 54  (Node Leaf 886  Leaf))
     14
     (Node (Node Leaf 982 Leaf) 246 (Node Leaf 3958 Leaf))
```

Is this what you expected? If you remove *return* () $\gg$ in the definition of *labelTree* and try again, what happens? What do these results imply?

3 (30%)**.** QuickCheck's *Arbitrary* class is defined as follows

```
class Arbitrary a where
  arbitrary :: Gen a
```

The type *Gen* is defined as

```
newtype Gen a = MkGen {unGen :: StdGen → Int → a}
```

(These definitions are from QuickCheck-2. The definitions in QuickCheck-1 are slightly different, but essentially the same. It does not matter which version you use.) Look at the QuickCheck source code for the definition of the monad instance. Assemble an equivalent monad from the *Reader* and *SplitReader* monads or monad transformers.

Define the function *sizedInt* :: *Gen Int* just using *ask*, *lift* and *System.Random.randomR* (i.e., not using the internal structure of the *Gen* type), such that *sizedInt* generates a random number between $-n$ and n where n is the read integer.