LiU-FP2010 Part II, Linkping, 17–19 May 2010

Exercises, Lectures 4 and 6: Monads

Henrik Nilsson

For the examination, do 1, 3, and 4.

1. Verify that `Maybe a` indeed is a monad by verifying the monad laws for `mbReturn` and `mbSeq`.

2. It turns out that many familiar data types in fact can be viewed as monads. For example, `[a]` can be understood as representing a computation with zero or more possible results ("nondeterminism"), and thus forms a monad with the appropriate definitions for `return` and `>>=`. Without "cheating" by looking ahead at the next lecture, show that `[a]` is a monad. *Hint:* `return` corresponds to a computation with exactly one result, while `>>=` needs to feed all possible outcomes form the first computation into the second, and then collect all possible results from that.

3. Below are the type signatures for a number of monad utility functions from the Haskell prelude and the module `Monad`. Define these utilities in terms of the basic monad operations. (If it is not reasonably clear from the type signatures what the intended meaning of each function is, ask!)

```
sequence  :: Monad m => [m a] -> m [a]
sequence_ :: Monad m => [m a] -> m ()
mapM      :: Monad m => (a -> m b) -> [a] -> m [b]
mapM_     :: Monad m => (a -> m b) -> [a] -> m ()
when      :: Monad m => Bool -> m () -> m ()
foldM     :: Monad m => (a -> b -> m a) -> a -> [b] -> m a
liftM     :: Monad m => (a -> b) -> (m a -> m b)
```

4. The Diagnostics monad `D` mentioned in the lectures represents computations that can emit error messages and, if necessary, give up completely and stop. Here is a variation of some of the operations on this monad:

Operation	Type	Purpose
`emitErrD`	`String -> D ()`	Emit an error message.
`failD`	`String -> D a`	Emit an error message and stop.
`failIfErrorsD`	`String -> D a`	Stop if one or more error messages have been emitted.
`stopD`	`D a`	Stop.
`runD`	`D a -> (Maybe a, [String])`	Run a diagnostic computation, returning any result and a list of all emitted error messages.

   - Think about what effects the diagnostics monad combine. For example, there is a standard notion of a *writer monad*:

     type W a = (a, T)

     for any type `T` that is a *monoid*: has an identity element and an associative binary operation. Such a monad is typically used for logging purposes. For example, `T` could be taken to be lists of error messages (strings), with list concatenation `++` as the binary operation to combine the output from sequentially composed computations and `[]` as the identity element. Would a writer monad be suitable for the logging part of the diagnostics monad as specified above, or is a more general notion of state needed? Why? *Hint:* Think about what information the various operations above need to have access to.

- Implement the diagnostics monad from scratch.
- Reimplement the diagnostics monad by using monad transformers to define the basic monad, and then defining the application-specific interface described above in terms of the standard operations for the monad obtained through the transformations.