Prof. Dr. Peter Thiemann
Janek Spaderna
janek.spaderna@pluto.uni-freiburg.de

Winter Term 2022/23
Issued 2023-01-30
Discussion on 2023-02-03

Functional Programming
https://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2022/

Exercise Sheet 12

1 Monad Transformers

The mtl package builds upon the transformers library used in last week's exercise sheet. Instead of programming against concrete a concrete transformer stack the programmer specifies the function's requirements in the type signature. The consumer is then able to choose the underlying monad stack themselves. We can extend last week's table by an "interface" column.

Standard Monad	Transformer	Interface	Base Type	Combined Type
`Maybe`	`MaybeT`		`Maybe a`	`m (Maybe a)`
`Either`	`ExceptT`	`MonadError`	`Either e a`	`m (Either e a)`
`Reader`	`ReaderT`	`MonadReader`	`r -> a`	`r -> m a`
`Writer`	`WriterT`	`MonadWriter`	`(a, w)`	`m (a, w)`
`State`	`StateT`	`MonadState`	`s -> (a, s)`	`s -> m (a, s)`
	`RWST`	`MonadRWS`	combines `Reader`, `Writer`, `State`	

Exercise 1 (File system state)

The goal of this exercise is to write a monad transformer which implements the `MonadState` interface by writing the state to the file system instead of, like `StateT`, keeping the state in the program memory and passing it from one action to the next.

1. Define a monad transformer **`FileStateT`** including the `run...` function. The path from which state is read/to which it is written should not be hardcoded but specified as an argument. Implement the customary typeclasses (e. g. **`Functor`**, ..., **`MonadIO`**, **`MonadTrans`**).
2. Implement the `MonadState` interface. All types which implement **`Read`** and **`Show`** should be supported.

   Note Due to Haskell's laziness you will have to use `System.IO.readFile'` for reading the file instead of `readFile` from the `Prelude`. A `get` would otherwise lock the file until the returned value has been forced. An expression such as `get <* put 1` would result in a run-time error.
3. Your transformer should support the other interfaces (**`MonadReader`**, etc.) if the transformed monad supports these. Write corresponding instances.
4. Write a function `memoizingFib :: MonadState (Map Integer Integer)` to calculate the n-th Fibonacci number. Use the **`MonadState`** constraint to memoize intermediate results.
5. Given `memoizingFib` two different interpretations. One using the ordinary state transformer, and one using this exercise's **`FileStateT`**. The latter should check if the file storage exists and initialize it to the empty map if not.

Table 1: Stack calculator operations

Operation	**Description**
`Noop`	Leaves the stack unchanged
`Pop`	Discards the stack's top element
`Push` *v*	Put's the value *v* on top of the stack
`Dup`	Duplicates the topmost value
`Dip` *p*	Executes *p* without the topmost value on the stack
`Swap`	Swaps the two topmost values
`Add` `Neg`	Performs the arithmetic operation
`LessEq`	Performs the comparison operation
`Not` `And`	Performs the logical operation
p1 `:&` *p2*	Sequences programs *p1* and *p2*
`If` *pT* *pF*	Executes *pT* if `True` is on top of the stack and *pF* otherwise
`While` *p*	Executes *p* repeatedly as long as it leaves `True` on top of the stack

2 GADTs

Exercise 2 (Type safe stack calculator)

We previously implemented a simple stack calculator. It only supported arithmetic operations and always returned `0` on underflow. We now want to extend it with logical operations and disallow programs which underflow the stack. Additionally, there should be no coercion between integer values and booleans.

1. Table 1 lists the operations we want to support. Define the data type `SProg` to represent programs consisting of these operations. Use a GADT to ensure type safety. More specifically, the state of the stack before and after the operation should be tracked in the type.
2. Implement a tag-free interpreter for `SProg`.
3. Define an expression `dup2` to duplicate the *two* values on top of the stack using only the basic operations from Table 1. Use it to write an `SProg` expression to calculate the maximum of two numbers. Write a property test.