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

Winter Term 2022/23
Issued 2022-12-04
Discussion on 2022-12-09

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

Exercise Sheet 7

To help you with writing and understanding your first monad instances download `SimplePrelude.hs` linked on the course homepage. It contains an alternative, simplified `Monad` typeclass. You can find usage instructions at the top of the file.

Exercise 1 (Non-determinism)

The goal of this exercise is to implement a solver for the N-Queens problem: how can n chess queens be placed on an $n \times n$ chess board, such that no queen can attack any other queen.

We will implement this using the list monad which allows for a straightforward, if not always very efficient, formulation of backtracking search. Start from `NQueens.hs` linked on the course homepage. It defines how solutions should be represented and functions to visualize them.

1. Extend `SimplePrelude.hs` with the `Monad` instance for lists. You can find the definition on the lecture slides.
2. Write a function `guard :: Bool -> [()]`. Its effect is a filter on the set of results. It should fulfill the following properties:

```haskell
prop_guardTrue, prop_guardFalse :: [Integer] -> [Integer] -> Bool
prop_guardTrue  xs ys = (xs >> guard True  >> ys) == (xs >> ys)
prop_guardFalse xs ys = (xs >> guard False >> ys) == []
```

   Note The module `Control.Monad` contains a more generic version of `guard`. It makes use of an abstraction we will learn about in a future lecture.

3. Implement the `nqueens` function using the list monad and backtracking search.
4. Verify that your solution is lazy enough; i. e. materializing only the first solution using `head (nqueens 11)` should run considerably faster than exhausting the complete search space with `length (nqueens 11)`.

Exercise 2 (This monad, that monad)

Values of type `Either a b` carry either a value of type `a` or a value of type `b`. The `These` type is related. Its values contain either an `a`, a `b`, or a combination of `a` and `b`:

```haskell
data These a b = This a | That b | These a b
```

It is available from the these package. However, the goal is to write the `Monad` instance yourself. For this, copy the definition above into your Haskell file rather than using the 'these' library.

1. Write the `Monad` instance for `These`. It should behave as a combination of generalized `Trace` and `Raise` effects, which were discussed in the last lecture.
2. Write an `Arbitrary` instance for `These`. You only have to provide the `arbitrary` function but implementing `shrink` can help by producing smaller counterexamples. If you wish to do so, `shrink` should derive a list "smaller" values from a given value by calling `shrink` on the subcomponents. You can make use of the list monad for this.
3. Test your `Monad` instance. Write three property tests, each verifying one of the monadic laws. Use values of type `These [Integer] Integer` for your tests.

Exercise 3 (Evaluation)

The file `MiniLang.hs` linked on the course homepage provides data types which model a small programming language. The goal of this exercise is to write an interpreter using the **`State`** monad.

The program's memory should be represented by a **`Map`** (from `Data.Map.Strict` in the containers library) from variable names of type **`Var`** to **`Integer`** values. All variables are initially zero. Results from boolean expressions have to be represented as numbers as well, à la C.

1. Extend `MiniLang.hs` with the definition of the **`State`** monad from the lecture. Additionally, write functions to retrieve and update variable values:

   ```
   getVar :: Var -> State Memory Integer
   setVar :: Var -> Integer -> State Memory ()
   ```

2. Write a functions to evaluate expressions, statements and whole programs. They should all make use of the **`State`** monad.

3. Write a function to run programs. Its type should be **`Prog -> Memory`**. Test your code; the file contains a program to sum the integers in a closed range $[a, b]$.