LVA 185.A05 Advanced Functional Programming (SS 2020)

Self-Assessment Test 4

Wednesday, 22 April 2020

Topics: Part IV, Chapters 12 and 13

Monads, Arrows

(No submission, no grading)

Part IV, Chapter 12 'Monads'

1. What are the monad laws? What do they require, ensure?
2. What does it mean to make a monad?
3. Can every type be made a monad? (Do not just answer 'yes' or 'no').
4. Rewrite equivalently using monadic operations:
   (a) `[ f x | x <- xs ]`
   (b) `[ x | x <- xs, p x ]`
5. The (standard definitions of the)
   (a) identity
   (b) list
   (c) maybe
   (d) map
   (e) state
   (f) input/output

   monad uses the default implementations of `return` and `fail` of `Monad`. Right or wrong? Give the actual implementation if a default implementation is not used.
6. Complete the instance declaration of the state monad:

```
newtype State st a = St (st -> (st,a))
instance Monad (State st) where
 (St h) >>= f = ...
 return x     = ...
 ... >> ...   = ...
 fail ...     = ...
```

7. Show that the defining equation of `(>@>)`:

```
(>@>) :: Monad m => (a -> m b) -> (b -> m c) -> (a -> m c)
f >@> g = \x -> (f x) >>= g
```

   is type correct, i.e., the right-hand side and the left-hand side expression of the defining equation of `(>@>)` have the same type.
8. Complete the instance declaration:

```
instance Monad (Either a) where
...
```

9. Is your `(Either a)` instance of `Monad` sound? Proof or counter-example.

10. If your `(Either a)` instance of `Monad` is not sound, can you revise the instance definition to make it sound? Prove soundness.
11. What is a monad-plus?
12. What are the monad-plus laws?
13. How is the list monad-plus defined?
14. Why is input/output a challenge for purely functional languages?
15. Why is the monad concept appealing for handling input/output in Haskell?

Part IV, Chapter 13 'Arrows'

1. What is an arrow? What is a sound arrow?
2. The below can be made arrows:
   (a) `Int`
   (b) `Bool`
   (c) `Char`
   (d) `Ordering`
   (e) `IO`
   (f) `[Int]`
   (g) `[a]`
   (h) `[]`
   (i) `Maybe Int`
   (j) `Maybe a`
   (k) `Maybe`
   (l) `Either Int`
   (m) `Either a`
   (n) `Either`
   (o) `(Int -> Int)`
   (p) `(a -> Int)`
   (q) `(Int -> b)`
   (r) `(a -> b)`
   (s) `(a ->)`
   (t) `(-> b)`
   (u) `(->)`

   Right or wrong? Meaningful or not? Why?
3. Implement an arrow instance of your choice. Prove for some laws that it is sound.
4. What is intuitively the meaning, the purpose of the arrow operations? What is their meaning for the map arrow?
5. Where is `Arrow` sitting in the type class hierarchy of Haskell'98? Where in more recent versions of Haskell?

Parts I – IV, Various Chapters

1. Haskell is an absolutely pure functional programming language. Do you agree? Why? Or, why not?
2. Dijkstra considered the 'go to' statement harmful. Why?
3. On that occasion: What is the given name of Dijkstra? What is he famous for? Name a few of his achievements.
4. Though not an established or widely used term, what is the Münchhausen principle? Give an example illustrating your answer.
5. Considering the current library modules `Data.Array` and `Data.Array.IO`, what do they support? How do they differ in the interface they offer a programmer?
6. The (default) list type in Haskell is not recommended as implementation type for abstract data types like queues and stacks. Why?
7. Explain the conceptual difference between an abstract and a concrete data type definition.
8. There are three meta (or: high level) goals that shall be accomplished with defining data types abstractly. Which ones?
9. What is a monoid? What is a sound monoid? Give an example.
10. Can stacks be made a monoid? Why? Or, why not?
11. Give a few examples of top-down algorithm patterns.
12. Give an example where we used wholemeal programming in this course.
13. Give the definition of the list functor.
14. Show that the defining equations of your list functor instance are type correct.
15. What is the meaning of the `fmap` operation of the map functor?
16. What is an applicative?
17. Show that the applicative law (AL4) is type correct:

    ```
    u <*> pure y  = pure ($ y) <*> u
    ```

18. What is the kind of an applicative?
19. Explain why prime number recognition could be dealt with as a functional pearl problem? Why were it a desaster, if someone were too successful in solving this pearl?
20. Prove by equational reasoning that

    ```
    f  :: Int -> Int -> Int
    f a b = (a + b)^2

    g :: Int -> Int -> Int
    g a b = a^2 + 2*a*b + b^2
    ```

    denote the same function.
21. Give a corecursive definition generating Pascal's triangle as a stream of lists.
22. Give an example which illustrates the generate/transform pattern of stream programming.
23. What algorithm pattern is related to memoization?
24. Define sets as an abstract data type in Haskell.
25. Why is it useful to have type class `Functor` in Haskell?