Problem Set 3

IAP 2020 18.S097: Programming with Categories

Due Friday January 31

Good academic practice is expected. In particular, cooperation is encouraged, but assignments must be written up alone, and collaborators and resources consulted must be acknowledged.

We suggest that you attempt all problems, but we do not expect all problems to be solved. We expect some to be easier if you have more experience with mathematics, and others if you have more experience with programming. The problems in this problem set are in general a step more difficult than those in previous problem sets. Nonetheless, the guideline remains that five problems is a good number to attempt, write up, and submit as homework.

Question 1. *Pullbacks and limits.*

Products and terminal objects are examples of limits. Another example is the pullback. Let $\mathcal{C}$ be a category, and let $X \xrightarrow{f} A \xleftarrow{g} Y$ be a pair of morphisms in $\mathcal{C}$. The *pullback* of $X \xrightarrow{f} A \xleftarrow{g} Y$ is an object $X \times_A Y$ and a pair of morphisms $\pi_1 \colon X \times_A Y \to X$ and $\pi_2 \colon X \times_A Y \to Y$ such that for all commuting squares

$$\begin{tikzcd}
C \arrow[r, "h"] \arrow[d, "k"'] & X \arrow[d, "f"] \\
Y \arrow[r, "g"'] & A
\end{tikzcd}$$

there is a unique map $u \colon C \to X \times_A Y$ such that

$$\begin{tikzcd}
C \arrow[dr, dashed, "u"] \arrow[drr, bend left, "h"] \arrow[ddr, bend right, "k"'] & & \\
& X \times_A Y \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] & X \arrow[d, "f"] \\
& Y \arrow[r, "g"'] & A
\end{tikzcd}$$

commutes.

We'll think about pullbacks in the category **Set** of sets and functions.

(a) What is the pullback of the diagram $\mathbb{N} \xrightarrow{!} \underline{1} \xleftarrow{!} \mathbb{B}$?

(b) What is the pullback of the diagram $X \xrightarrow{!} \underline{1} \xleftarrow{!} Y$?

(c) What is the pullback of the diagram $\mathbb{N} \xrightarrow{\text{isEven}} \mathbb{B} \xleftarrow{\texttt{yes}} \underline{1}$? Here $\mathbb{N}$ is the set of natural numbers $\mathbb{B} = \{\texttt{yes}, \texttt{no}\}$, and $\text{isEven}\colon \mathbb{N} \to \mathbb{B}$ answers the question "Is n even?".

(d) What is the pullback of the diagram $\mathbb{N} \xrightarrow{\text{isEven}} \mathbb{B} \xleftarrow{\text{isOdd}} \mathbb{N}$? Here isOdd answers the question "Is n odd?".

(e) Given a general description of the pullback of some diagram $X \xrightarrow{f} A \xleftarrow{g} Y$ in **Set**.

(f) How is this notion similar to that of terminal objects and products?

Question 2. *Catamorphisms.*

Consider the functor `F`:

```
data F a = Nil | Cons Int a
  deriving Functor
```

We define the recursive type:

```
type ListInt = Fix F
```

(a) Let `isEven :: Int -> Bool` take an integer to `True` if it is even, and `False` otherwise. Here is an `F`-algebra.

```
hello :: F Bool -> Bool
hello Nil = False
hello Cons n a = isEven n || a
```

What is the induced catamorphism `cata hello :: ListInt -> Bool`?

(b) Implement the function `product :: ListInt -> Int` that takes a list of integers and returns their product.

Question 3. *Naturals.*

Natural numbers can be represented in Haskell as a recursive data structure

```
data Nat = Zero | Succ Nat
```

(a) Implement a type `Nat2`, isomorphic to `Nat`, but this time defined as an initial algebra of a functor.

(b) Using a catamorphism, define a function `Nat2 -> Int` that maps n to the n-th Fibonacci number.

(c) Define a coalgebra whose anamorphism is a (partial) function `Int -> Nat2` that sends a non-negative `Int` into its fixed-point representation (ie. its representation as a value of `Nat2`).

Question 4. *Merge sort.*

A hylomorphism is the composite of an anamorphism and a catamorphism – the idea is that the anamorphism unfolds a data structure, and the catamorphism refolds it in some convenient way. One application of this idea is to the implementation of sorting algorithms.

In this question, we implement merge sort using a hylomorphism. Here's the idea: The seed (the carrier of the coalgebra) is the list to be sorted. Use this function

```
split :: [a] -> ([a], [a])
split (a: b: t) = (a: t1, b: t2)
  where
    (t1, t2) = split t
split l = (l, [])
```

to split the list into two lists and use them as new seeds. Make sure you know how to deal with empty lists.

The carrier of the algebra is again a list (this time it's actually a sorted list, but this cannot be reflected in the type). Your partial results are sorted lists. You combine them using this function.

```
merge :: Ord a => [a] -> [a] -> [a]
merge (a: as) (b: bs) =
  if a <= b
  then a : merge as (b: bs)
  else b : merge (a: as) bs
merge as [] = as
merge [] bs = bs
```

Make sure your program also works for empty lists (it should return an empty list).

Question 5. *Monoids as List algebras.*

Recall the monad List: $\mathbf{Set} \to \mathbf{Set}$ that sends a set X to the set of lists of elements of X, whose multiplication flattens a list of lists, and whose unit takes an element $x \in X$ to the singleton list $[x] \in \mathrm{List}(X)$.

(a) Given a List-algebra $a\colon \mathrm{List}(X) \to X$, construct a monoid on the set X.
(b) Given a monoid $(X, *, e)$, construct a list algebra.
(c) Show that your two constructions are inverses (we hope they are!).

Question 6. *Hello world.*

Using the `IO` monad, write a program that prints `Hello world!`.

Question 7. *The tree monad.*

Recall that a binary tree with leaves valued in A is an element of the initial F-algebra where F is the functor:

$$\begin{aligned} F_A\colon \mathbf{Set} &\longrightarrow \mathbf{Set}; \\ X &\longrightarrow A + X \times X; \\ (f\colon X \to Y) &\longmapsto \big((\mathrm{id}_A + f \times f)\colon A + X \times X \to A + Y \times Y\big) \end{aligned}$$

(a) Define a monad $T\colon \mathbf{Set} \to \mathbf{Set}$ that maps a set A to the set of trees with leaves valued in A.
(b) Implement this monad in Haskell.

Question 8. *Adjunctions and monads.*

Adjunctions are very closely related to monads. In fact, every adjunction induces a monad by composing the adjoint functors, and every monad factors as the composite of adjoints (in possibly many different ways). Let's explore this a little.

Before we begin, let's recall the notion of an algebra for a monad. Given a monad (T, μ, η) on a category $\mathcal{C}$, recall that a T-algebra (X, a) is a morphism $a\colon TX \to X$ such that the diagrams

$$\begin{tikzcd} X \arrow[r, "a"] \arrow[dr, "\mathrm{id}_X"'] & TX \arrow[d, "a"] \\ & X \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} TTX \arrow[r, "Ta"] \arrow[d, "\mu_X"'] & TX \arrow[d, "a"] \\ TX \arrow[r, "a"'] & X \end{tikzcd}$$

Given algebras (X, a) and (Y, b), a morphism between them is a morphism $f\colon X \to Y$ in $\mathcal{C}$ such that

$$\begin{tikzcd} TX \arrow[r, "a"] \arrow[d, "Tf"'] & X \arrow[d, "f"] \\ TY \arrow[r, "b"'] & Y \end{tikzcd}$$

commutes. Algebras and their morphisms form a category $T\mathbf{Alg}$.

(a) Given a monad (T, μ, η) on $\mathcal{C}$ and an object A in $\mathcal{C}$, show that (TA, μ_A) is an algebra for T. We call this the *free algebra on* A.
(b) Show that for every monad (T, μ, η) on $\mathcal{C}$ there is an adjunction

$$\begin{tikzcd} \mathcal{C} \arrow[r, shift left, "L"] & T\mathbf{Alg} \arrow[l, shift left, "R"] \end{tikzcd}$$

where L maps an object A of $\mathcal{C}$ to its free algebra (TA, μ_A), and R maps a T-algebra (X, a) to its carrier object X.
(c) Suppose that we have an adjunction

$$\begin{tikzcd} \mathcal{C} \arrow[r, shift left, "L"] & \mathcal{D} \arrow[l, shift left, "R"] \end{tikzcd}$$

with L the left adjoint and R the right adjoint. Show that there is a monad with underlying functor $(L \fatsemi R)\colon \mathcal{C} \to \mathcal{C}$.

Question 9. *The continuation monad.*

For any set S, the S-*continuations monad* $C_S\colon \mathbf{Set} \to \mathbf{Set}$ has the following functor part: $X \mapsto (S^{S^X})$, or in haskell

```
data Cont s x = Cont ((x -> s) -> s)
```

Do each of the following either in Haskell notation or in mathematical notation: your choice.

(a) Either write `fmap` for this functor, or say in mathematical notation how C_S acts on morphisms.
(b) Give the return `return x :: x -> Cont s x` for this monad, i.e. $\eta\colon \mathrm{id}_{\mathbf{Set}} \to C_S$.
(c) Give the join for this monad `join :: Cont s (Cont s x) -> Cont s x`.
(d) Consider the Kleisli morphisms, and use currying to see them as "continuations". What is being "continued"?

Question 10. *The Yoneda embedding (Challenge!).*

An important theorem in category theory is the Yoneda embedding. Here we imagine the Yoneda embedding as a game. First, we choose a category $\mathcal{C}$ together that we both completely understand. Then I pick a secret object a in $\mathcal{C}$. Your goal is to find an object that's isomorphic to my secret object.

You're allowed to get information about the object in two ways. First, if you name an object x, I must tell you the set $y_a(x) = \mathcal{C}(x,a)$ abstractly as a set, but I don't have to tell you anything about how its elements are related to the morphisms you see in $\mathcal{C}$. Second, if you name a morphism $m\colon x \to x'$, I have to tell me the function $y_a(m)\colon \mathcal{C}(x',a) \to \mathcal{C}(x,a)$ between those abstract sets. The Yoneda lemma says that you can always win this game.

Give a strategy for winning the game in a general category $\mathcal{C}$. (You can assume $\mathcal{C}$ has finitely many objects and morphisms if you want.)

Question 11. *Tell a story.*

You run into a math major friend in the infinite corridor, and they ask about this course. In approximately half a page, explain to them something interesting you learned.

Question 12. *Grade the pset.*

Give a grade to this problem set, taking into account how much you learned, how interesting or fun it was, and how much time you spent on it. Explain your grade.