Problem Set 1 Solutions

IAP 2020 18.S097: Programming with Categories

Question 1. *Functions in mathematics and Haskell.* Suppose $f\colon \texttt{Int} \to \texttt{Int}$ sends an integer to its square, $f(x) \coloneqq x^2$ and that $g\colon \texttt{Int} \to \texttt{Int}$ sends an integer to its successor, $g(x) \coloneqq x + 1$.

(a) Write f and g in Haskell, including their type signature and their implementation.
(b) Let $h \coloneqq f \circ g$. What is $h(2)$?
(c) Let $i \coloneqq f \fatsemi g$. What is $i(2)$?

Our Solution 1.

(a)
```
f :: Int -> Int
g :: Int -> Int
f x = x * x      -- x^2 also works
g x = x + 1      -- succ x also works
```

(b) $h(2) = f(g(2)) = g(2) * g(2) = 3 * 3 = 9$

(c) $i(2) = g(f(2)) = f(2) + 1 = 4 + 1 = 5$

Question 2. *A tiny category.*

Recall that a *category* consists of the data:
1. a set $\mathrm{Ob}(\mathcal{C})$ of objects;
2. for every pair of objects $c, d \in \mathrm{Ob}(\mathcal{C})$ a set $\mathcal{C}(c, d)$ of morphisms;
3. for every three objects b, c, d and morphisms $f\colon b \to c$ and $g\colon c \to d$, a specified morphism $(f \fatsemi g)\colon b \to d$ called the *composite of f and g*;
4. for every object c, an identity morphism $\mathrm{id}_c \in \mathcal{C}(c, c)$; and

subject to two laws:

Unit: for any $f\colon c \to d$, the equations $\mathrm{id}_c \fatsemi f = f$ and $f \fatsemi \mathrm{id}_d = f$ hold.

Associative: for any $f_1\colon c_1 \to c_2$, $f_2\colon c_2 \to c_3$, and $f_3\colon c_3 \to c_4$, the equation $(f_1 \fatsemi f_2) \fatsemi f_3 = f_1 \fatsemi (f_2 \fatsemi f_3)$ holds.

This tiny category is sometimes called the *walking arrow category* **2**.

$$\mathbf{2} \coloneqq \boxed{\mathrm{id}_1 \circlearrowright \overset{1}{\bullet} \xrightarrow{f} \overset{2}{\bullet} \circlearrowright \mathrm{id}_2}$$

(a) Write down the set of objects, the four sets of morphisms, the composition rule, and the identity morphisms.
(b) Prove that this category obeys the unit and associative laws.

Our Solution 2.

We'll write $\mathcal{C}$ instead of **2** just to make things a little clearer.

(a) Objects: $\mathrm{Ob}(\mathcal{C}) = \{1, 2\}$.
Morphisms:

$$\mathcal{C}(1,1) = \{\mathrm{id}_1\},$$
$$\mathcal{C}(1,2) = \{f\},$$
$$\mathcal{C}(2,1) = \varnothing,$$
$$\mathcal{C}(2,2) = \{\mathrm{id}_2\}.$$

Composition:

$$f \circ \mathrm{id}_1 = f$$
$$\mathrm{id}_2 \circ f = f,$$
$$\mathrm{id}_1 \circ \mathrm{id}_1 = \mathrm{id}_1$$
$$\mathrm{id}_2 \circ \mathrm{id}_2 = \mathrm{id}_2.$$

Identities: $\mathrm{id}_1, \mathrm{id}_2$.

(b) One sees above that composing with the identity really is unital: every time an id is composed with any morphism, the result is that morphism. For associativity, we need to check five things:

$$(\mathrm{id}_1 \circ \mathrm{id}_1) \circ \mathrm{id}_1 =^{?} \mathrm{id}_1 \circ (\mathrm{id}_1 \circ \mathrm{id}_1)$$
$$(f \circ \mathrm{id}_1) \circ \mathrm{id}_1 =^{?} f \circ (\mathrm{id}_1 \circ \mathrm{id}_1)$$
$$(\mathrm{id}_2 \circ f) \circ \mathrm{id}_1 =^{?} \mathrm{id}_2 \circ (f \circ \mathrm{id}_1)$$
$$(\mathrm{id}_2 \circ \mathrm{id}_2) \circ f =^{?} \mathrm{id}_2 \circ (\mathrm{id}_2 \circ f)$$
$$(\mathrm{id}_2 \circ \mathrm{id}_2) \circ \mathrm{id}_2 =^{?} \mathrm{id}_2 \circ (\mathrm{id}_2 \circ \mathrm{id}_2)$$

They all are true! Both sides are id_1, f, f, f, and id_2 respectively.

In general, if we have checked the unitality condition, then we don't need to check the associativity condition for any composable triples f, g, h that include the identity: i.e. if any of f, g, or h is an identity then the equation $(f \circ g) \circ h = f \circ (g \circ h)$ automatically holds. This would be a nice exercise for those who want to think about it, and it would obviate the need to do any work for part b. of this exercise!

Question 3. *Is it an isomorphism?.* Suppose that someone tells you that their category $\mathcal{C}$ has two objects c, d and two non-identity morphisms, $f\colon c \to d$ and $g\colon d \to c$, but no other morphisms. Does f have to be the inverse of g, i.e. is it forced by the category axioms that $g \circ f = \mathrm{id}_c$ and $f \circ g = \mathrm{id}_d$?

Our Solution 3. Yes. Since the types are $f\colon c \to d$ and $g\colon d \to c$, the type of $g \circ f$

must be $c \to c$, and there is only one morphism in $\mathcal{C}(c,c)$, namely id_c. This means $g \circ f = \mathrm{id}_c$, and similarly we get that $f \circ g = \mathrm{id}_d$. These two equations are the very definition of f being the inverse of g, i.e. of these being isomorphisms.

Question 4. *Almost categories.*

(a) Give an example of some data – objects, morphisms, composition, and identities – that satisfies the associative laws but not the unit law.
(b) Give an example of some data – objects, morphisms, composition, and identities – that satisfies the unit laws but not the associative law.

Our Solution 4. We'll give two examples in each case.

(a) *Example 1:* Let $\mathcal{C}$ be the category with one object, $\mathrm{Ob}(\mathcal{C}) = \{\bullet\}$ and the homset $\mathcal{C}(\bullet,\bullet) = \mathbb{Z}_{>0} = \{1,2,3,\dots\}$ – the set of all positive whole numbers. We compose two morphisms by adding them: $f \circ g = f + g$. No choice of identity morphism will allow us to satisfy the unit law; for example, if we were to choose 1, we would find that $1 \circ 2 = 1 + 2 = 3$, which is not equal to 2 like the unit law requires.

Example 2: Let $\mathcal{C}$ be the category with one object, $\mathrm{Ob}(\mathcal{C}) = \{\bullet\}$ and two morphisms, say $A, N \colon \bullet \to \bullet$ (we're thinking of the words "All" and "None"). Say $f \circ g = A$ for all f, g. Then this certainly satisfies the associative law: $(f \circ g) \circ h = A$ for all morphisms f, g, h. But it does not satisfy the unit law, because there is no morphism f for which $f \circ N = N$: neither A nor N work.

(b) *Example 1:* Let $\mathcal{C}$ be the category with one object, $\mathrm{Ob}(\mathcal{C}) = \{\bullet\}$ and the homset $\mathcal{C}(\bullet,\bullet) = \mathbb{N} = \{0,1,2,3,\dots\}$ – the set of all non-negative whole numbers. We compose two morphisms by taking the absolute value of their difference: $f \circ g = |f - g|$. The identity is given by 0, since for every number f we have $|0 - f| = |f - 0| = f$. This, however, is not associative; for example

$$1 \circ (2 \circ 3) = 1 \circ |2 - 3| = 1 \circ 1 = |1 - 1| = 0,$$

whereas

$$(1 \circ 2) \circ 3 = |1 - 2| \circ 3 = 1 \circ 3 = |1 - 3| = 2.$$

Example 2: Let $\mathcal{C}$ be the category with one object, $\mathrm{Ob}(\mathcal{C}) = \{\bullet\}$ and four different morphisms, $\mathcal{C}(\bullet,\bullet) = \{\mathrm{id}, f, g, h\}$. For a composition law, first say that id composed with anything is itself, so we're guaranteed that the unit law holds. It remains to violate the associative law.

$$\begin{array}{lll} f \circ f = f, & f \circ g = f, & f \circ h = g, \\ g \circ f = f, & g \circ g = f, & g \circ h = f, \\ h \circ f = f, & h \circ g = f, & h \circ h = f \end{array}$$

Then $f \circ (g \circ h) = f \circ f = f \neq g = f \circ h = (f \circ g) \circ h$.

Question 5. *Monoids.*

A *monoid* $(M, *, e)$ is
1. a set M;
2. a function $*\colon M \times M \to M$; and
3. an element $e \in M$ called the *identity*;

subject to two laws:

Unit: the equations $e * m = m$ and $m * e = m$ hold for any $m \in M$.

Associative: the equation $(m_1 * m_2) * m_3 = m_1 * (m_2 * m_3)$ holds for any $m_1, m_2, m_3 \in M$.

(a) Show that $(\mathbb{N}, +, 0)$ forms a monoid.

(b) A string in 0 and 1 is a (possibly) empty sequence of 0s and 1s; examples include 0, 11, 0110, 0101110 and so on. We write the empty string []. Let $\text{List}_{\{0,1\}}$ be the set of strings in 0 and 1. Given two strings a and b, we may concatenate them to form a new string ab. Show that $\text{List}_{\{0,1\}}$, together with concatenation and the empty string [], form a monoid.

(c) Explain why (prove that) every monoid can be viewed as a category with a single object.

Our Solution 5.

(a) We have a set $\mathbb{N}$, we have a function $+\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and an element $0 \in \mathbb{N}$; we just need to check that the laws hold. The unit laws are that $0 + m = m$ and $m + 0 = m$ for any natural number $m \in \mathbb{N}$; that's true! And similarly, $(m_1 + m_2) + m_3 = m_1 + (m_2 + m_3)$ for any three natural numbers.

(b) The set is $\text{List}_{\{0,1\}}$, which we'll denote L. The function ${+}{+}\colon L \times L \to L$ is concatenation, e.g. $[0,1,1] {+}{+} [1,1,0,1,0] = [0,1,1,1,1,0,1,0]$. More formally, suppose that ℓ is a list of length m and ℓ' is a list of length m'. Write $\ell = [\ell_0, \ldots, \ell_{m-1}]$ and $\ell' = [\ell'_0, \ldots, \ell'_{m'-1}]$. Their concatenation is given by the formula

$$\ell {+}{+} \ell' = [\ell_0, \ldots, \ell_{m-1}, \ell'_0, \ldots, \ell'_{m'-1}].$$

The empty list [ ] is the unit. Clearly concatenating a list ℓ with the empty list does not change the list: $\ell {+}{+} [\,] = \ell$ and $[\,] {+}{+} \ell = \ell$. The associative law is also easy: both $(\ell {+}{+} \ell') {+}{+} \ell''$ and $\ell {+}{+} (\ell' {+}{+} \ell'')$ are given by

$$[\ell_0, \ldots, \ell_{m-1}, \ell'_0, \ldots, \ell'_{m'-1}, \ell''_0, \ldots, \ell''_{m''-1}].$$

(c) The definition of category in question 2 and the definition of monoid in this question look remarkably similar. If a category $\mathcal{C}$ has one object, say $\bullet$, then there is only one set of morphisms, namely $\mathcal{C}(\bullet, \bullet)$. Calling that set M, we indeed have a set of morphisms. The composition formula in $\mathcal{C}$ becomes the function necessary for the monoid; the identity for the unique object becomes the unit of the monoid. The unit and associativity laws for the category translate precisely into the unit and associativity laws for the monoid.

Question 6. *Preorders.*

> A *preorder* is a category such that, for every two objects a, b, there is at most one morphism $a \to b$. That is, there either is or is not a morphism from a to b, but there are never two morphisms a to b. If there is a morphism $a \to b$, we write $a \leq b$; if there is not a morphism $a \to b$, we don't.

For example, there is a preorder $\mathcal{P}$ whose objects are the positive integers $\mathrm{Ob}(\mathcal{P}) = \mathbb{N}_{\geq 1}$ and whose hom-sets are given by

$$\mathcal{P}(a, b) \coloneqq \{x \in \mathbb{N} \mid x * a = b\}$$

This is a preorder because either $\mathcal{P}(a, b)$ is empty (if b is not divisible by a) or contains exactly one element.

(a) What is the identity on 12?
(b) Show that if $x \colon a \to b$ and $y \colon b \to c$ are morphisms, then there is a morphism $y \circ x$ to serve as their composite.
(c) Would it have worked just as well to take $\mathcal{P}$ to have all of $\mathbb{N}$ as objects, rather than just the positive integers?

Our Solution 6.

(a) 1 is the identity on 12. It's an element—in fact the unique element—of the hom-set $\mathcal{P}(12, 12) = \{x \in \mathbb{N} \mid x * 12 = 12\}$.

(b) If $x \colon a \to b$ is a morphism then $x * a = b$. If $y \colon b \to c$ is a morphism then $y * b = c$. Then $(y*x)*a = y*(x*a) = y*b = c$, so we have found an element $y*x \in \mathcal{P}(a, c)$. So we take $y \circ x \coloneqq y * x$.

(c) Yes and no. With the exact definition given for $\mathcal{P}(a, b)$ above, the answer is no. In order to have a preorder, there must be at most one morphism between any two objects a, b, and this is violated for $a = b = 0$. Indeed, $\{x \in \mathbb{N} \mid x*0 = 0\} = \mathbb{N}$ has more than one element. But the answer is yes if we slightly modify the definition of $\mathcal{P}(a, b)$ to have a special case when both a and b are 0:

$$\mathcal{P}(a, b) = \begin{cases} \{x \in \mathbb{N} \mid x * a = b\} & \text{if } a > 0 \text{ or } b > 0 \\ \{1\} & \text{if } a = 0 \text{ and } b = 0 \end{cases}$$

Here one can check that there is at most one morphism between any two objects. The only cases to check are $a = 0, b > 0$ and $a > 0, b = 0$ and $a = 0, b = 0$. In the first case we have $\mathcal{P}(a, b) = \{x \mid x * 0 > 0\} = \varnothing$; that checks out. In the second case, when $a > 0$ we have $\mathcal{P}(a, b) = \{x \mid x * a = 0\} = \{0\}$, which has one element; that checks out. And in the third case when $a = b = 0$ we have defined $\mathcal{P}(a, b) = \{1\}$, which has one element, so that also checks out. It works!

Question 7. *Church Booleans.* Boolean logic may be encoded in the lambda calculus

using the following definitions:

$$\begin{aligned}\text{True} &= \lambda x.(\lambda y.x)\\ \text{False} &= \lambda x.(\lambda y.y)\\ \text{AND} &= \lambda p.(\lambda q.(pq)p)\\ \text{OR} &= \lambda p.(\lambda q.(pp)q)\end{aligned}$$

Evaluate the lambda terms (AND True) False and (OR False) True.

Our Solution 7. First we'll show everything in gory detail.

In the first subproblem will rename variables in False to avoid some ambiguity. So let's use False $= \lambda x'.(\lambda y'.y')$.

$$\begin{aligned}\text{(AND True) False} &= \Big(\lambda p.\big(\lambda q.(pq)p\big)\big(\lambda x.(\lambda y.x)\big)\Big)\big(\lambda x'.(\lambda y'.y')\big)\\ &= \lambda q.\Big(\big((\lambda x.(\lambda y.x))q\big)\big(\lambda x.(\lambda y.x)\big)\Big)\big(\lambda x'.(\lambda y'.y')\big)\\ &= \Big(\big(\lambda x.(\lambda y.x)\big)\big(\lambda x'.(\lambda y'.y')\big)\Big)\big(\lambda x.(\lambda y.x)\big)\\ &= \big(\lambda y.\big(\lambda x'.(\lambda y'.y')\big)\big)\big(\lambda x.(\lambda y.x)\big)\\ &= \lambda x'.(\lambda y'.y') = \text{False}\end{aligned}$$

In the second subproblem, we repeat False, so we'll rename its variables twice.

$$\begin{aligned}\text{(OR False) True} &= \Big(\lambda p.\big(\lambda q.(pp)q\big)\big(\lambda x.(\lambda y.y)\big)\Big)\big(\lambda x.(\lambda y.x)\big)\\ &= \Big(\lambda q.\Big(\big(\lambda x'.(\lambda y'.y')\big)\big(\lambda x''.(\lambda y''.y'')\big)\Big)q\Big)\big(\lambda x.(\lambda y.x)\big)\\ &= \Big(\big(\lambda x'.(\lambda y'.y')\big)\big(\lambda x''.(\lambda y''.y'')\big)\Big)\big(\lambda x.(\lambda y.x)\big)\\ &= (\lambda y'.y')\big(\lambda x.(\lambda y.x)\big)\\ &= \lambda x.(\lambda y.x) = \text{True}\end{aligned}$$

Wow, I can't believe it works!

But there's an easier way to do both: just keep True and False around.

$$\begin{aligned}\text{(AND True) False} &= \lambda p.(\lambda q.(pq)p))\ \text{True False}\\ &= (\lambda q.(\text{True } q))\ \text{True}\\ &= \text{True False True}\\ &= (\lambda x.(\lambda y.x))\ \text{True False}\\ &= (\lambda y.\text{False})\ \text{True}\\ &= \text{False}\end{aligned}$$

Similarly,

$$\begin{aligned}\text{(OR False) True} &= (\lambda p.(\lambda a.(pp)q))\ \text{True False}\\ &= (\lambda q.(\text{False False})q)\ \text{True}\\ &= \text{False False True}\\ &= (\lambda x.(\lambda y.y))\ \text{False True}\\ &= (\lambda y.y)\ \text{True}\\ &= \text{True}\end{aligned}$$

Question 8. *The Y combinator.* The Y combinator is an iconic lambda term whose reduction does not terminate. It is defined as follows:

$$Y = \lambda f.\big((\lambda x.f(xx))(\lambda x.f(xx))\big)$$

Compute Yg.

Our Solution 8.

$$\begin{aligned} Yg &= \lambda f.\big((\lambda x.f(xx))(\lambda x.f(xx))\big)g \\ &= (\lambda x.g(xx))(\lambda x.g(xx)) \\ &= g\big((\lambda x.g(xx))(\lambda x.g(xx))\big). \end{aligned}$$

We will continue on making the required substitutions, but we pause here to note the equation above. Ok, continuing on...

$$\begin{aligned} Yg &= g\big((\lambda x.g(xx))(\lambda x.g(xx))\big) && \text{above equation} \\ &= g\big(g((\lambda x.g(xx))(\lambda x.g(xx)))\big) && \text{substitution for } x \\ &= g(Yg) && \text{above equation.} \end{aligned}$$

In fact, it keeps going on and on, $Yg = g(Yg) = g(g(Yg))$ and so on. So Yg is called a fixed point of g.

Question 9. *Defining a toy category in Haskell.* In Haskell, we may define a typeclass whose instances are the data of a category in the following way:

```
class Category obj mor | mor -> obj where
  dom :: mor -> obj
  cod :: mor -> obj
  idy :: obj -> mor
  cmp :: mor -> mor -> Maybe mor
```

The "Maybe" part is saying that two morphisms may not compose (e.g. $f\colon a \to b$ and $g\colon b' \to c$ only compose if $b = b'$). Note also that there are no laws—associative or unital—so the user of this class has to just certify that these laws really do hold in their specific case. Finally, the weird `| mor -> obj` thing at the top is called a "functional dependency" and is just there to soothe the compiler.

To compile this typeclass definition, three language pragmas need to be enabled. This can be done by including the following code at the top of the file:

```
{-# language FlexibleInstances #-}:
{-# language MultiParamTypeClasses #-}
{-# language FunctionalDependencies #-}
```

(a) Implement the category $\mathbf{2} = \boxed{\bullet \to \bullet}$ from Question 2 as an instance of this `Category` class.

Our Solution 9.

```
data Ob = Ob0 | Ob1
data Mor = Id0 | M01 | Id1

mydom :: Mor -> Ob
mydom Id0 = Ob0
mydom M01 = Ob0
mydom Id1 = Ob1

mycod :: Mor -> Ob
mycod Id0 = Ob0
mycod M01 = Ob1
mycod Id1 = Ob1

myidy :: Ob -> Mor
myidy Ob0 = Id0
myidy Ob1 = Id1

mycmp :: Mor -> Mor -> Maybe Mor
mycmp Id0 Id0 = Just Id0
mycmp Id0 M01 = Just M01
mycmp Id0 Id1 = Nothing
mycmp M01 Id0 = Nothing
mycmp M01 M01 = Nothing
mycmp M01 Id1 = Just M01
mycmp Id1 Id0 = Nothing
mycmp Id1 M01 = Nothing
mycmp Id1 Id1 = Just Id1

instance Category Ob Mor where
  dom = mydom
  cod = mycod
  idy = myidy
  cmp = mycmp
```

Question 10. *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.

Our Solution 10. You guys did great!