Programming with Categories - Problem Set 1

Alex Klibisz - aklibisz@gmail.com

January 21, 2019

Original questions are here: http://brendanfong.com/programmingcats_files/ps1.pdf

Question 1

```haskell
f :: Int -> Int
f x = x * x

g :: Int -> Int
g x = x + 1

h :: Int -> Int
h x = f (g x)

i :: Int -> Int
i x = g (f x)

main = do
    print (h 2) -- apply g, then f to get 9
    print (i 2) -- apply f, then g to get 5
```

$h := f \circ g$ means to apply g first, then f, which is $f(g(2)) = 9$.

$i := f; g$ means to apply f, then g, which is $g(f(2)) = 5$

Question 2

a) Objects, morphisms, composition, and identity

1. The objects are $Ob(\mathcal{C}) = \{1, 2\}$.
2. The morphisms are: $\mathcal{C}(1,1) = id_1$, $\mathcal{C}(1,2) = \{f\}$, $\mathcal{C}(2,1) = \emptyset$, $\mathcal{C}(2,2) = \{id_2\}$.

3. Composition is defined: $id_1; f = f; id_2 = f$.
4. Identity morphisms are $id_1 : 1 \to 1$ and $id_2 : 2 \to 2$.

b) Proof for unit and associative laws

Unit laws $f : 1 \to 2$ is the only non-identity morphism. According to composition, $id_1; f = f$ and $f; id_2 = f$, satisfying the unit laws.

Associative The only sequence of three non-identity morphisms is id_1, f, id_2. So we can check the two possible associative arrangement for equality:

- $(id_1; f); id_2 = f; id_2 = f$, using composition to substitute $(id_1; f) = f$.
- $id_1; (f; id_2) = id_1; f = f$, using composition to substitute $(f; id_2) = f$.

Question 3

Yes; In order for $\mathcal{C}$ to be a category, there must exist composites of morphisms f and g. For $f : c \to d$ and $g : d \to c$, there are two composites: $f; g : c \to c$ and $g; f : d \to d$. We assume two identity morphisms: $id_c : c \to c$ and $id_d : d \to d$. As stated in the question, f and g are the only non-identity morphisms. Since there can't be any other morphisms $c \to c$ and $d \to d$, $f; g : c \to c$ must be $id_c : c \to c$ and $g; f : d \to d$ must be $id_d : d \to d$.

Question 4

a) An *almost category* satisfying associative laws but not unit law

- Single object is a
- Morphisms are the natural numbers $\mathbb{N} = 1, 2, ...$
- Identity morphism $id_a = 1$
- Composition of morphisms $f_1; f_2 = min(f_1, f_2)$.

Associative laws hold because

- $(f_1; f_2); f_3 = f_1; (f_2, f_3) = f_1$ for $f_1 < f_2 < f_3$
- $(f_1; f_2); f_3 = f_1; (f_2, f_3) = f_2$ for $f_1 > f_2 < f_3$
- $(f_1; f_2); f_3 = f_1; (f_2, f_3) = f_3$ for $f_1 > f_2 > f_3$

Unit law is violated because

- $id_a; f_1 = f_1; id_a = id_a = 1$ since f_1 can be no less than 2.

b) An *almost category* satisfying unit laws but not associative

- Single object is a.
- Three non-identity morphisms: *rock, paper, scissors*, abbreviated r, p, s.
- Identity morphism $id_a = \mathit{forfeit}$, abbreviated f.
- Composition is defined
  - $r; p = p; r = p$ (paper beats rock).
  - $r; s = s; r = r$ (rock beats scissors).
  - $s; p = p; s = s$ (scissors beats paper).
  - $f; m = m; f = m, \forall m \in \{r, p, s\}$ (anything beats forfeit).

Unit law holds because any morphism $f; m = m; f = m, \forall m \in r, p, s$.

Associative laws are violated because $(r; s); p = r; p = p \neq r; (s; p) = r; s = r$.

The idea was inspired by this response on the Math StackExchange.

Question 5

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

This forms a monoid because it satisfies the unit and associative laws:

- Unit: $0 + n = n + 0 = n \forall n \in \mathbb{N}$.
- Associative: $(0 + n_1) + n_2 = n_1 + n_2 = 0 + (n_1 + n_2) \forall n \in \mathbb{N}$.

b) Show that $(\mathbf{List}_{0,1}, ++, [])$ forms a monoid.

Note: ++ denotes string concatenation.

This forms a monoid because it satisfies the unit and associative laws:

- Unit: $[] + +s = s + +[] = s \forall s in \mathrm{List}_{0,1}$.
- Associative: $(s_1 + +s_2) + +s_3 = s_1 + +(s_2 + +s_3) = s_1 + +s_2 + +s_3 \forall s_i in \mathrm{List}_{0,1}$

c) Prove that every monoid is a category with a single object

Enumerate the translation from a monoid $(M, *, e)$ to a single-object category $\mathcal{C}$.

- $Ob(\mathcal{C}) = \{1\}$.

- Morphisms $\mathcal{C}(e, e) = M$.
- Identity morphism e from the set M.
- Composition for any $m_1 : a \to b$, $m_2 : b \to c$ is $m_3 : a \to c = m_1; m_2 = m_1 * m_2$.

Then show the unit and associative laws hold.

- Unit: The definition for a monoid states that $e * m = m * e = m$. Consequently, $id_1; m = e * m = m = m; id_1 = m * e$.
- Associative: The definition for a monoid states the $(m_1 * m_2) * m_3 = m_1 * (m_2 * m_3)$ holds for any three morphisms $m_1, m_2, m_3 \in M$.

Question 6

Pre-order $\mathcal{P}$ with objects $Ob(\mathcal{P}) = \mathbb{N}_{\geq 1}$ and morphisms $\mathcal{P}(a, b) := \{x \in \mathbb{N} | x * a = b\}$.

a) Identity on 12

Identity for 12 is 1, because this is the only element $x \in Ob(\mathcal{P})$ satisfying $x * 12 = 12$.

b) There exists a composite $y \circ x$ for morphisms $x : a \to b$ and $y : b \to c$

The morphisms $x : a \to b$ and $y : b \to c$ can be expressed as $x : r * a = b$ and $y : s * b = c$ for some $r, s \in \mathbb{N}_{\geq 1}$.

If there exists some $t \in \mathbb{N}_{\geq 1}$ such that $t * a = c$, then the composite $y \circ x : a \to c$ exists. Solve for t to show it exists.

$$\begin{aligned} t * a &= c \\ t * \frac{b}{r} &= s * b \\ t &= s * b * \frac{r}{b} \\ t &= r * s \end{aligned} \tag{1}$$

c) Does $\mathbb{N}_{\geq 0}$ also form a preorder?

No; there is exactly one value of x satisfying $x * a = b, \forall (a, b) \in \mathbb{N}_{\geq 1}$, and therefore exactly one morphism. For $a = 0 \neq b$ and $a \neq 0 = b$ there exists no values of x and therefore no morphisms; this is still a valid a preorder. However, for $a = b = 0$, there are infinitely many satisfying values of x and therefore infinitely many morphisms. This violates the requirement that there must be either zero or one morphisms for any (a, b).

Question 7

Note: using Haskell-style pseudocode. '\' corresponds to λ.

```
AND T F
= ((\p.(\q.(pq)p))(T))(F) -- Replace p with T in the expression (\q.(pq)p).
= (\q.(T q) T)            -- Replace q with F in the expression ((T q) T).
= (T F) T                 -- Replace T with its definition.
= (\x.(\y.x) F) T         -- Replace x with F in the expression (\y.x).
= (\y F) T                -- Expression (\y F) takes parameter T, always returns F.
= F
```

```
OR F T
= ((\p.(\q.(pp)q)) F) T -- Replace p with F in the expression (\q.(pp)q)
= (\q.(F.F)q)           -- Replace q with T in the expression ((F.F)q)
= (F.F) T               -- Replace F with its definition.
= (\x.(\y.y) F) T       -- Replace x with F in the expression (\y.y).
= (\y.y) T              -- Replace y with T.
= T
```

Question 8

```
Y g
= (\f.((\x.f(xx))(\x.f(xx)))) g -- Replace f with g in the inner terms f(xx).
= (\x.g(xx)(\x.g(xx))           -- Replace x with (\x.g(xx)) in the inner term g(xx).
= g((\x.g(xx)) (\x.g(xx)))      -- Replace x with (\x.g(xx)) in the inner term g(xx).
= g(g((\x.g(xx)) (\x.g(xx))))   -- Recursion ensues..
= g(g(g((\x.g(xx)) (\x.g(xx)))))
= g(g(g ... (Y g)))
```

Question 9

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


-- Typeclass whose instances are categories. Cateogires have objects and morphisms.
-- Morphisms each have an input object, the domain, and an output object, the codomain.
class Category obj mor | mor -> obj where
    -- Given a morphism you can get its input object.
    dom :: mor -> obj
    -- Given a morphism you can get its output object.
    cod :: mor -> obj
    -- Given an object you can get its identity morphism.
    idy :: obj -> mor
    -- Two morphisms might not compose: f: a -> b and g: b' -> c compose iff b = b'.
    cmp :: mor -> mor -> Maybe mor

data Object = One | Two deriving (Show, Eq)
data Morphism = Morphism Object Object deriving (Show, Eq)

instance Category Object Morphism where
    dom (Morphism a b) = a
    cod (Morphism a b) = b
    idy a = Morphism a a
    cmp (Morphism One One) (Morphism One Two) = Just (Morphism One Two)
    cmp (Morphism One Two) (Morphism Two Two) = Just (Morphism Two Two)
    cmp m1 m2
        | m1 == m2 = Just m1
        | otherwise = Nothing

main = do
    print (dom (Morphism One Two)) -- "One"
    print (cod (Morphism One One)) -- "One"
    print (cod (Morphism Two Two)) -- "Two"
    -- TODO: why do you have type annotation for idy?
    print (idy One :: Morphism) -- "Morphism One One"
    print (idy Two :: Morphism) -- "Morphism Two Two"
    print (cmp (Morphism One One) (Morphism One Two)) -- "Just (Morphism One Two)"
    print (cmp (Morphism One One) (Morphism Two Two)) -- "Nothing"
```