18.S097 PS2 (in Swift)

Jasdev Singh

January 25, 2020

Acknowledgements.

Tomás Ruiz-López helped me get a better sense of Bow's higher-kinded-type emulation in answering 2b.

Question 1.

Rephrased, Question 1 is asking for the cardinality of the objects in the functor category, $[\mathbf{Set}, \mathbf{3}]$, i.e. $|\mathbf{Obj}([\mathbf{Set}, \mathbf{3}])|$. To start, we have three functors into $\mathbf{3}$ at hand $K_1, K_2, K_3 \in \mathbf{Obj}([\mathbf{Set}, \mathbf{3}])$ (borrowing notation from Question 2).

In asking if there are any functors beyond these, we need to make an observation about **Set**'s hom-sets. $\forall S_1, S_2 \in \mathbf{Obj}(\mathbf{Set}), S_1 \neq \emptyset, S_2 \neq \emptyset : \mathrm{Hom}(S_1, S_2) \neq \emptyset$. Home-sets in **Set** are non-empty for all *non-empty* set pairings. There will always be functions between them. Now, let's turn our attention to the empty set. Its self-hom-set only contains the identity morphism, $\mathrm{id}_\emptyset$. And all of its originating hom-sets—$\mathrm{Hom}(\emptyset, S)$ for $S \in \mathbf{Obj}(\mathbf{Set}), S \neq \emptyset$—are empty since there we can't construct functions from non-empty sets into the empty set.

We can lean on the above to make sure connections (morphisms) aren't broken across our functors.

Let's make this more precise.

The remaining functors need to map $\emptyset$ to an object in $\mathbf{3}$ that *only* has outbound morphisms. Then, all other objects in **Set** must be mapped in the usual way. Listing out the steps for our next candidate functor F_1,

(a) Map $\emptyset$ and $\mathrm{id}_\emptyset$ to 1 and id_1, respectively.

(b) Map all $\emptyset$-originating morphisms to a.

(c) Map all non-empty sets, their identities, and morphisms between non-empty sets to 2 and id_2 (akin to K_2).

We have some degrees of freedom in carrying out (a)–(c). That is, we can form another functor F_2 by, substituting in 2 for all mentions of 1 in (a). b for a in (b). And 3 for all occurrences of 2 in (c).

Similarly, to form an F_3. Leave (a) as is. $b \circ a$ for a in (b). And 3 for all occurrences of 2 in (c).

$K_1, K_2, K_3, F_1, F_2, F_3$ exhaust all possible structure preserving mappings between **Set** and **3**.

Question 2.

(a) K_B preserves

- compositions, since $\forall S_1, S_2, S_3 \in \mathbf{Obj}(\mathbf{Set})$ and $\forall f : S_1 \to S_2, \forall g : S_2 \to S_3, K_B(g \circ f) = \mathrm{id}_B = \mathrm{id}_B \circ \mathrm{id}_B = K_B(g) \circ K_B(f)$.
- identities, since $\forall S \in \mathbf{Obj}(\mathbf{Set}), K_B(\mathrm{id}_S) = \mathrm{id}_B = \mathrm{id}_{K_B(S)}$.

(b) Mirroring Bow's approach to higher-kinded-type emulation and with `boolConst` representing the constant functor on Swift's `Bool` type.

```
class Kind<F, A> {
        init() {}
}

final class ForConst {}
final class ConstPartial<Constant>: Kind<ForConst, Constant> {}
typealias ConstOf<Constant, Tag> = Kind<ConstPartial<Constant>, Tag>

final class Const<Constant, Tag>: ConstOf<Constant, Tag> {
        let constant: Constant

        init(_ constant: Constant) {
                self.constant = constant
        }

        static func fix(
                _ fConstant: ConstOf<Constant, Tag>
        ) -> Const<Constant, Tag> {
                fConstant as! Const<Constant, Tag>
        }
}

postfix operator ^
postfix func ^<Constant, Tag>(
        _ fConstant: ConstOf<Constant, Tag>
) -> Const<Constant, Tag> {
        Const.fix(fConstant)
}
```

```
protocol Functor {
        static func map<A, B>(
                _ fA: Kind<Self, A>,
                _ transform: (A) -> B
        ) -> Kind<Self, B>
}

extension ConstPartial: Functor {
        static func map<OldTag, NewTag>(
                _ fA: ConstOf<Constant, OldTag>,
                _ transform: (OldTag) -> NewTag
        ) -> ConstOf<Constant, NewTag> {
                Const(fA^.constant)
        }
}

func boolConst<A>(_ constant: Bool) -> Const<Bool, A> {
        Const(constant)
}
```

Question 3.

(a) For δ to define a natural transformation, the following diagram must commute ($\forall f : X \to Y$ in $\mathrm{Hom}(X, Y)$, with $X, Y \in \mathbf{Obj}(\mathbf{Set})$).

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\delta_X"] & Y \arrow[d, "\delta_Y"] \\
X \times X \arrow[r, "\mathrm{Double}(f)"] & Y \times Y
\end{tikzcd}$$

Following the bottom path for an arbitrary $x \in X$,

$$x \xmapsto{\delta_X} (x, x) \xmapsto{\mathrm{Double}(f)} (f(x), f(x))$$

And the upper,

$$x \xmapsto{f} f(x) \xmapsto{\delta_Y} (f(x), f(x))$$

Both paths are equivalent! δ is indeed a natural transformation between $\mathrm{id}_{\mathbf{Set}}$ and Double.

(b)
```
func diag<A>(_ a: A) -> (A, A) {
        (a, a)
}
```

Question 4.

Question 5.

Question 6.

Question 7.

Question 8.