The Haskell Programmer's Guide to the IO Monad
– Don't Panic –
第 2 回

今井宜洋

平成 20 年 4 月 11 日

概 要

この文章は [] の 4 章を読んで、まとめたものである、圏論特有の用語に慣れるため、本名に加え、日本語訳として使われる言葉をカギカッコ「」で付記した。この文章達に、間違い、もしくは不適切な点があれば、教えていただけると幸いである。

4 章　Functors「関手」

4.1 functor「関手」とは

category 同士の対応を考えよう。もし、category の構造を変えない、いい対応なら、それを functor とよぼう。

定義 1
$\mathcal{A}, \mathcal{B}$ がそれぞれ *category*「圏」であるとする。ここで、F が、*category* $\mathcal{A}$ から $\mathcal{B}$ への *functor*「関手」であるとは、$\mathcal{O}_{\mathcal{A}}$ から $\mathcal{O}_{\mathcal{B}}$ への写像 $F_{\mathcal{O}}$ と、$\mathcal{M}_{\mathcal{A}}$ から $\mathcal{M}_{\mathcal{A}}$ への写像が定まり、かつ、次の 3 つのことが成り立つことである。

1.
$$\forall f : A \underset{\mathcal{A}}{\to} B, \quad F_{\mathcal{M}}(f) : F_{\mathcal{O}}(A) \underset{\mathcal{B}}{\to} F_{\mathcal{O}}(B)$$

2.
$$\forall A \in \mathcal{O}_{\mathcal{A}}, \quad F_{\mathcal{M}}(id_A) = id_{F_{\mathcal{O}}(A)}$$

3.
$$\forall f : A \underset{\mathcal{A}}{\to} B, \ \forall g : B \underset{\mathcal{A}}{\to} C, \quad F_{\mathcal{M}}(g \circ_{\mathcal{A}} f) = F_{\mathcal{M}}(G) \circ_{\mathcal{B}} F_{\mathcal{M}}(f)$$

今日はこの *3* つの項目を覚えて帰ろう。

functor の二つの写像 $F_{\mathcal{O}}$, $F_{\mathcal{M}}$ の小さな $_{\mathcal{O},\mathcal{M}}$ は、省略されることもあるよ。

定義 2
functor $F : \mathcal{A} \to \mathcal{A}$ のように、同じ *category* 間の *functor* のことを特別に *endofunctor* と呼んだりもする。

functor 同士の composit や、恒等的な functor を考えるのは簡単だよ。次みたいな感じで定義しよう。

定義 3
categories $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ と、*functors* $F : \mathcal{A} \to \mathcal{B}$, $G : \mathcal{B} \to \mathcal{A}$ に対して、*category* $\mathcal{A}$ 上の *identities*「恒等関手」$I : \mathcal{A} \to \mathcal{A}$ を次で定義する。

$$I_{\mathcal{O}}(A) := A$$

$$I_{\mathcal{M}}(f) := f$$

また、二つの *functor* $F : \mathcal{A} \to \mathcal{B}$, $G : \mathcal{B} \to \mathcal{C}$ に対して、*composition* GF を次で定義する。

$$\forall A \in \mathcal{O},\ (GF)(A) := G(F(A))$$

$$\forall f \in \mathcal{M},\ (GF)(f) := G(F(f))$$

この定義によると、カッコを省略して $GF(A)$ とか $GF(f)$ とか書いても曖昧ではない。おんなじ functor F を何回も composite するときは、F^n みたいな書き方も許すとしよう。

練習 1

- *identity* I は *functor* である。
- 二つの *functors* F, G の *composition* GF は、再び *functor* である。

練習

4.2 Functors implement structure

category の定義では object の構造そのものには言及しなかった。functor は、object にある種の構造を付加するツールの役割になる。
[略]

4.3 Haskell category $\mathcal{H}$ における functor

Haskell category $\mathcal{H}$ 上の endofunctor $F : \mathcal{H} \to \mathcal{H}$ を考えよう。object(型) における写像 $F_{\mathcal{O}}$ を一引数の型コンストラクタ、morphism(関数) における写像 $F_{\mathcal{M}}$ を fmap という一つの写像で定めよう。

型コンストラクタは

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

それぞれの functor で、この `fmap` をいい感じに overloading して使う。

例として Maybe 構造及び, List 構造を考えよう。この、２つの class は上記の `Functor` クラスのインスタンスである。

- Maybe について、object における写像、morphism における写像をそれぞれ次のように定める。

$$\mathtt{t} \in \mathcal{O}_{\mathcal{H}},\ Maybe_{\mathcal{O}}(\mathtt{t}) = \mathtt{Maybe\ t}$$

$$\mathtt{f} \in \mathcal{M}_{\mathcal{H}},\ Maybe_{\mathcal{M}}(\mathtt{f}) = \mathtt{fmap\ f}$$

  ただし、ここでの `fmap` は次で定める。

```
fmap :: (a -> b) -> Maybe a -> Myabe b
fmap f Nothing = Nothing
fmap f (Just x) = Just (f x)
```

- List について、object における写像、morphism における写像をそれぞれ次のように定める。

$$List_{\mathcal{O}}(\mathtt{t}) = [\mathtt{t}]$$

$$List_{\mathcal{M}}(\mathtt{f}) = \mathtt{fmap\ f}$$

  ただし、ここでの `fmap` は次で定める。

```
fmap :: (a -> b) -> [a] -> [b]
fmap f [] = []
fmap f (x:xs) = (f x):(fmap f xs)
```

補題 1
Maybe 及び、*List* は両方とも *functor* になる。

[証明]:
functor の 3 条件のうち、1 つ目は O.K. なので、2 及び 3 つ目を考えよう。そ

の際、morphism のイコールを示したくなるが、$\mathcal{H}$ における morphism のイコールは、次のような感じになることに注意しよう。

$$f = g \iff \forall x : X, f(x) = g(x)$$

[証明デモ]

■

まとめ

出てきた用語	Haskell の言葉
functor「関手」	型コンストラクタと fmap
identity「恒等関手」	id