

SYDNEY TYPE THEORY MEETUP - 24/07/17

Polymorphic functions & naturality - Richard Garner (richard.garner@mq.edu.au)

In Haskell, we can write functions of type $\forall a. [a] \rightarrow [a]$:

- reverse :: forall a. [a] -> [a]
 - tail
 - $\lambda_ \rightarrow []$
 - id
 - cycle left, cycle right
 - $h [] = []$
 - $h [x] = [x, x]$
 - $h [x, y] = [y, x, y]$
 - $h ls = ls ++ reverse ls$
- + composition

Non-examples

- sort :: $\forall a. \text{Ord } a \Rightarrow [a] \rightarrow [a]$
- nub :: $\forall a. \text{Eq } a \Rightarrow [a] \rightarrow [a]$

Problem: classify all functions of type $\forall a. [a] \rightarrow [a]$.

To solve: work with idealised version of Haskell without laziness, and with only strict, total functions.

Now use some category theory! Basic idea:

Polymorphism = naturality.

In this particular case, "naturality" means the following:

If $\theta :: \forall a. [a] \longrightarrow [a]$, and $f :: A \longrightarrow B$, then whenever $ls :: [A]$, we have

$$\underbrace{\text{instantiation of } \theta \text{ at the type } A}_{\text{eg:}} \quad (\text{map } f) (\theta_A ls) = \theta_B ((\text{map } f) ls) \quad (*)$$

$$=$$

In categorical lingo:-

- We have a category Hask whose objects are (non-lazy) Haskell types, and whose morphisms $A \longrightarrow B$ are (strict, total) Haskell functions of type $A \rightarrow B$ (identified up to extensional equivalence).
- We have a functor

$$[] : \text{Hask} \longrightarrow \text{Hask}$$

$$A \longmapsto [A]$$

$$f : A \rightarrow B \longmapsto \text{map } f : [A] \longrightarrow [B]$$

(satisfying axioms).

In Haskell: Class Functor f where

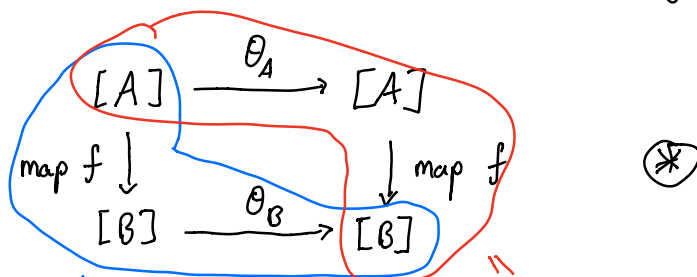
$$\text{fmap} :: (a \rightarrow b) \rightarrow (f a) \rightarrow f b$$

and $\{ [] \}$ is an instance of this.

$$\left\{ \begin{array}{l} \text{Maybe} \\ \text{IO} \\ \text{any monad} \end{array} \right\}$$

- A natural transformation $\theta : [] \Rightarrow []$ comprises:-

- for each $A \in \underline{\text{Hask}}$, a morphism $\theta_A : [A] \rightarrow [A]$ (components)
- such that for any $f: A \rightarrow B$ in $\underline{\text{Hask}}$, the following composites coincide:



ie: $\theta_B \circ (\text{map } f) = (\text{map } f) \circ \theta_A$

eg: reverse. $(\text{map } f) \circ \theta_B = \theta_A \circ (\text{map } f)$.

Idea from category theory:

polymorphic fns of type $\forall a. [a] \rightarrow [a]$
are the same as natural transformations $[] \Rightarrow []$.

so to classify the former, we may as well classify the latter.
 And we can do this!

Suppose we're given $\theta: [] \Rightarrow []$. We want to know how the component/installation $\theta_A: [A] \rightarrow [A]$ acts on some given list $[x_0, \dots, x_{n-1}]$.

Consider a type

Data Enum_n = 0 | 1 | ... | n-1.

There's a function $f: \text{Enum}_n \rightarrow A$ defined by

$$\begin{aligned} f \ 0 &= x_0 \\ &\vdots \\ f \ (n-1) &= x_{n-1} \end{aligned}$$

and now $[x_0, \dots, x_{n-1}] = (\text{map } f) [0, \dots, n-1]$.

So now we can consider

$$\begin{array}{ccc} [\text{Enum}_n] & \xrightarrow{\theta_{\text{Enum}_n}} & [\text{Enum}_n] \\ \text{map } f \downarrow & & \downarrow \text{map } f \\ [A] & \xrightarrow{\theta_A} & [A] \end{array}$$

Tracing $[0, \dots, n-1]$ in $[\text{Enum}_n]$ around this square, we get:

$$\begin{array}{ccc} [0, \dots, n-1] & \xrightarrow{\quad} & l_s \\ \downarrow & & \searrow \\ [x_0, \dots, x_{n-1}] & \xrightarrow{\quad} & \theta_A [x_0, \dots, x_{n-1}] \end{array}$$

Now, $l_s = [i_0, \dots, i_k]$ where each i is in $0, \dots, n-1$

So $(\text{map } f) l_s = [x_{i_0}, \dots, x_{i_k}]$.

In summary: if $\theta_{\text{Enum}_n} [0, \dots, n-1] = [i_0, \dots, i_k]$

then $\theta_A [x_0, \dots, x_{n-1}] = [x_{i_0}, \dots, x_{i_k}]$

(This argument is an instance of the Yoneda lemma in category theory)

What this means: to give $\theta: [] \Rightarrow []$

is equally to give, for each natural number n , a

list $l_n :: [\text{Enum}_n]$. We can encode this data as

(certain) Haskell functions $\text{Int} \rightarrow [\text{Int}]$.

... and we can do this in practice! (See example.hs)