

NICCOLÒ
VENTURELLO

6

28/01/2021

1.1

2) First part

(\rightarrow) is the ~~array~~ arrow type. It takes two parameter types a and b , and represents a function $(a \rightarrow b)$

28/01/2021

3) continued

executes f to obtain an mb

3.1

Monads are more flexible than applicatives
consider the following functions

3.2

if $M :: m \text{ Bool} \rightarrow m a \rightarrow m a \rightarrow m a$

~~if $M c t e = do$~~

if $M c t e = do$

$v \leftarrow c$

if v then t else e

3.3

if $A :: a \text{ Bool} \rightarrow a p \rightarrow a p \rightarrow a p$

if $A c t e = \text{pure } (\lambda c t e \rightarrow \text{if } c \text{ then } t \text{ else } e)$
 $\langle * \rangle c \langle * \rangle t \langle * \rangle e$

if we consider the type Maybe we have

if $M (\text{Just True}) (\text{Just } 1) \text{ Nothing} = \text{Just } 1$

but also

if $A (\text{Just True}) (\text{Just } 1) \text{ Nothing} = \text{Nothing}$

is less flexible [↗] if one parameter is Nothing
then all is Nothing

NICOLA
VERDARA
28/01/2021

3

3) continued

$$fmap: (a \rightarrow b \rightarrow c \rightarrow d) \rightarrow f a \rightarrow f b \rightarrow f c \rightarrow f d$$

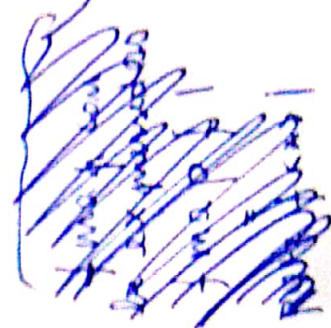
the applicative is characterized by

$$pure :: a \rightarrow qa$$

which boxes a value into an applicative
and by

$$\langle \star \rangle :: q(a \rightarrow b) \rightarrow qa \rightarrow qb$$

which abstracts the application



Monads are a more flexible extension of applicatives, and here

$$return :: a \rightarrow ma$$

same purpose of pure

$$(\gg=) :: ma \rightarrow (\underbrace{a \rightarrow mb}) \rightarrow mb$$

which binds the value contained in ma into f and

extension of applicative

* $fmap$ can be expressed in terms of $pure$ and $\langle \star \rangle$

$$fmap f = pure f \langle \star \rangle$$

28/01/2021

5.1

3)

A functor is a generalization of a type on which a function can be mapped.

It's characterized by a function

$$fmap :: (a \rightarrow b) \rightarrow f a \rightarrow f b$$

which maps a function with one parameter on the elements of the functor. For example, the list type is an instance of functor with

$$fmap = map$$

An applicative is a generalization of this concept, where we don't restrict ourselves to have a function with only one parameter. In a sense an applicative represent the following "generalization" of $fmap$

$$fmap :: a \rightarrow f a$$

$$fmap :: (a \rightarrow b) \rightarrow f a \rightarrow f b$$

NICCOLO'
VEITORELLO

28/01/2021

1

1) Thesis

~~add n m = add m n~~ $\text{add } n \ m = \text{add } m \ n$

BC $n = \text{Zero}$

$\text{add } \text{Zero } m = m$ by definition of add

6.1

$\text{add } m \ \text{Zero} = m$ by property 1
OK

IC $n = \text{Succ } x$

by definition of add

$\text{add } (\text{Succ } x) \ m = \text{Succ } (\text{add } x \ m)$

6.2

$= \text{Succ } (\text{add } m \ x) = \text{add } m \ (\text{Succ } m)$

by inductive hypothesis

by property 2

OK

Indice dei commenti

- 1.1 non rispondi sul kind
- 2.1 -> b
- 2.2 la conclusione è giusta, ma la prova non è completa
- 3.1 what does it mean, flexible?
- 3.2 I don't understand this
- 3.3 you describe an original idea, but more that about Applicative and Monad, your idea is that the conditional can be done in strict (ifA) and non-strict (ifM) versions. Also the if of (ifM) could be written with $\lambda c' t' e' \rightarrow \text{if } c' \text{ then } t' \text{ else } e' \rangle \vee t' e$. Unfortunately, this is not the relation between Applicative and Monad that we discussed in the course
- 4.1 you don't explain how these 2 functions are able to model all fmap_i
- 5.1 Functor is a type class that contains types of kind $* \rightarrow *$
- 6.1 $m = \text{add } m \text{ Zero}$
- 6.2 x