Monade in računski učinki

Računski učinki

$f : \text{int} \longrightarrow \text{int}$

- ↳ print / input
- ↳ throw
- ↳ stanje

$f : \mathbb{Z} \to \mathbb{Z}$

$\tan : \mathbb{R} \longrightarrow \mathbb{R}$

$\tan\left(\frac{\pi}{2}\right)$ ni definiran

$\tan : \{x \in \mathbb{R} \mid \neg \exists k \in \mathbb{Z}.\ x = \frac{\pi}{2} + k.\pi\} \to \mathbb{R}$

Program lahko sproži računske učinke.

Primeri:

- izjeme
- stanje
- vhod / izhod
- nedeterminizem: program lahko izbira med večimi možnostmi (npr. nedeterm. Turingov stroj)
- verjetnostno računanje: program izbira med možnostmi z verjetnostno porazdelitvijo
- kvantno računanje: superpozicija

Monada

- matematični koncept iz 1950'ih, teorija kategorij
  - ↳ posplošitev algebre: grupa, kolobar, vektorski prostor
  - ↳ in še več
- E. Moggi 1984: monade zajemajo računske učinke
- → Haskell prevzame monade kot programski konstrukt za opis računskih učinkov

Čiste vrednosti (pure value) ali vrednost

- koda brez računskih učinkov
- rezultat

Izračun (computation)

- koda, ki lahko ima računske učinke
- ko jo izvedemo, se lahko zgodijo učinki

V Haskellu:

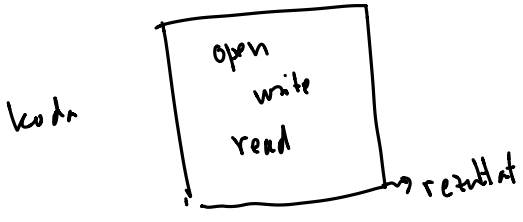
- imamo vrednosti: 42
- izračuni: učinki v Haskellu
  - izjema / napaka
  - divergenca (koda se nikoli ne konča, se zacikla)

Izračune predstavimo s tipi.

Primeri:

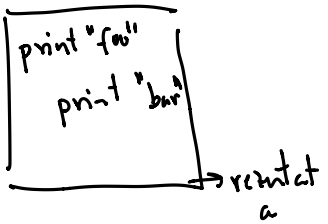
① Imamo ali nimamo rezultat: `Maybe a = Nothing` ← ni rezultata
`| Just a` ← rezultat je
↑ tip rezultata

② Napaka ali izjema: ko odpremo datetoko



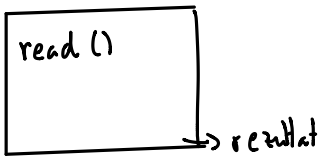

```
FileOperation a = Result a
                | FileNotFound
                | DiskFull
                | PermissionDenied
                ⋮
```

③ Output:



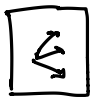

```
Output a = (String, a)
           ↑       ↑
           kaj se je   rezultat
           izpisalo
```

④ Input:



$$Input\ a = String \to a$$

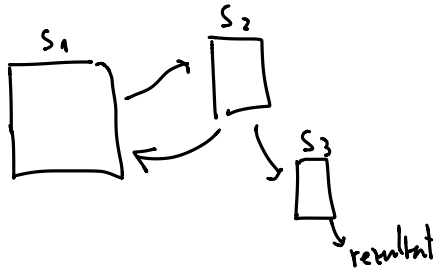
⑤ Nedeterminizem: več možnosti (nič ali več možnosti)



$$Nondeterministic\ a = [a]$$

↑ Seznam možnih rezultatov

⑥ Stanje: Stanja tipa s ↑ tip možnih stanj



$$State\ a = (s \to (s, a))$$

s ↑ začetno stanje; s ↑ končno stanje; a ↑ rezultat

Nadaljna struktura:

① Čisti izračuni: ča imamo čisto vrednost, jo lahko pretvorimo v čisti izračun

Primeri

① Manjkajoče vrednosti: $Just\ v$ imamo vrednost

② Output: $("", v)$ nicesar ne izpišemo

③ Input: $(\lambda x \to v)$ vhod ignoriramo

④ Nedeterminizem: $[v]$ imamo en rezultat

⑤ Stanje: $(\lambda x \to (x, v))$ stanje pustimo tako, kot je

② Kako komponiramo izračune

Intermezzo: izpit

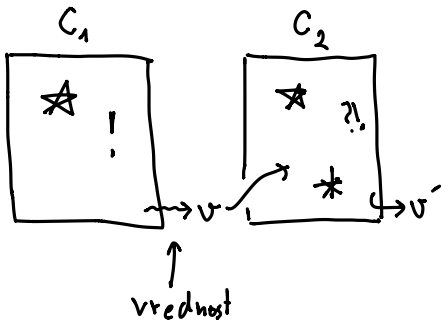
① Izpit bo na računalniku
- vprašanja na papirju

1. več krajših vprašanj z odgovori a, b, c, d
2. } prolog /dokazovanje pravilnosti/
3. } deklarativno prog.

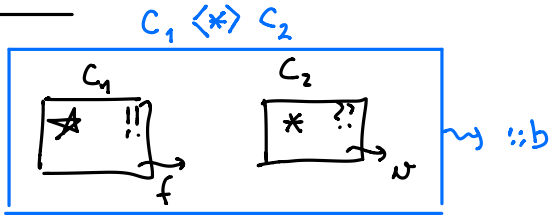
+110 točk, 100 točk je max.

② • odgovori na spletno učilnico

③ • lahko imate poljubne zapiske (lahko papir, lahko USB)

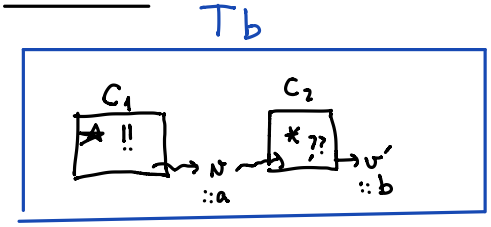


Applicative



Kako uporabimo funkcijo na vrednosti v prisotnosti učinkov

Monad



$c_1 :: Ta$ $\quad c_2 :: a \to Tb$

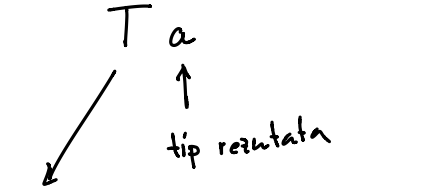
bind c_1 c_2

$c_1 >>= c_2$

najprej naredimo c_1 → nato c_2

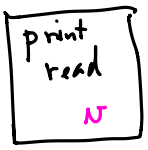
"do ..." (kasneje)

V splošnem imamo tip izračunov



Typeclass Functor:

T a → izračun "vsebuje" učinke & vrednosti



C_1

fmap f C_1

"Čisto funkcijo f uporabi na vrednostih, ki so "vsebovane v C_1"

$v :: a$

$f :: a \to b$

$f\ v :: b$

"Do"

$$C_1 >>= (\lambda x \to C_2) \quad \Rightarrow \quad \text{do } x \leftarrow C_1 \; C_2$$

$$C_1 >>= (\lambda x \to (C_2 >>= (\lambda y \to \ldots.)))$$

$$\Downarrow$$

```
do x ← C1
   y ← C2
   ....
```