

LAB 9 : POLYMORPHISM & CLASE

Recap: TYPE VARIABLES



în locul unui tip concret (Char, Int...)
! tip. mare

Ex:

```
*Main> :t map  
map :: (a -> b) -> [a] -> [b]
```

funcție

listă returnată: listă

a, b [variabile de tip

[pot fi înlocuite cu + tip.

POLYMORPHISM = mecanism Haskell prin care

se pot defini seturi de operații care

se comportă diferit în funcție de tip.

! 2 tipuri ↴

PARAMETRIC

AD-HOC

① POLYMORPHISM PARAMETRIC

- tipul conține una sau mai multe variabile de tip, dar fără constrainte
- ne ajută să definim "stocuri" / structuri de date generic

Ex: head :: [a] → a

fst :: (a, b) → a

[] :: [a]

(:) :: a → [a] → [a]

au ac. implementare pt. + tip

⇒ nu se poate să redifinim funcția pt tipuri diferite (utilizare de cod)

! Variabilele de tip sau valori în fcti
de context:

Ex: head :: [a] → a
 { [Int] → Int
 { [Bool] → Bool

! O xareahila' de h̄i care apare de mai
multi ori' n̄ Imlocuirele de aelasi h̄i
peste tot.

Ex: $\text{id} : a \rightarrow a$ [OK: $\text{id} : \text{Int} \rightarrow \text{Int}$ WRONG: $\text{id} : \text{Bool} \rightarrow \text{Int}$]

② POLIMORFISM AD-HOC

→ Sfârșitul de implementare diferit
pentru tipuri diferențiate
(→ apar contrignerile de tip)

Ex: Sfârșitul elem

- Tipul sfârșitului : $\text{elem_} :: \text{Eq } a \Rightarrow a \rightarrow [a] \rightarrow \text{Bool}$

contrignerile de tip

=> Sfârșitul se aplică pe liste cu elemente care defineste operatorul ($= =$)

- Implementare :

```
elem\_ []      = False
elem\_ x (y:ys) = x == y || elem\_ x ys
```

↓
de aici apare contragreșirea

Oby :

1)

→ elem - 2 [1, 3, 2, 4] \Rightarrow True

Dar, pentru tipul de listă
[data Color = Red / Green / Blue]

elem - Red [Blue, Red, Green]

✗

Eroare deoarece ($= =$) nu e definit

pentru Color

2)

($= =$) : are comportamentul diferit

în funcție de tipul pe care îl primește

\Rightarrow implementările diferă în funcție de tip

Comparare ($1 == 2$) și diferență de
("Hello" == "Hello!")

\Rightarrow implementarão diferentes comportamentos

similares

$$(=) : (\underbrace{\text{Eq } \alpha}_{\text{constrangendo } \alpha}) \rightarrow \alpha \rightarrow \alpha \rightarrow \text{Bool}$$

TYPE CLASSES & CLASS INSTANCES

TYPE CLASSES

[definește un comportament
~ interfață

Eq

[interfață pentru a verifica egalitatea
+ tip pentru care vom să
verificăm egalitatea poate fi instanță
a clasei Eq.

TYPE CLASS DEFINITION

```
class Eq a where
```

```
    (==), (/=) :: a -> a -> Bool
```

```
    x /= y = not (x == y)
```

-> membru implicit

```
    x == y = not (x /= y)
```

-> membru implicit

→ are funcții care vor fi implementate
de instanțele ei

→ membre 'simplicii' = fcti care au o implementare default în clasa

INSTANȚIATION

→ pentru ca un tip să fie instanță a unei clase, trebuie să fie implementate

Ex:

Eq

- dăte tipurile de bază și instanțează
- orică alt tip trebuie adăugat

```
data Person = Person {firstName :: String, lastName :: String} deriving Show  
p1 = Person "Buddy" "Finklestein"  
p2 = Person {firstName = "Buddy", lastName = "Finklestein"}  
p3 = Person "Guy" "Smith"
```

instance Eq Person where

```
person1 == person2 = firstName person1 == firstName person2 &&  
                     lastName person1 == lastName person2  
-- person1 /= person2 = not (person1 == person2) -- optional
```

Oky : Ve pot adauga si afișuri de date generice mult-o clasa:

Ex:

```
data BST a = Empty | Node a (BST a) (BST a)

instance Eq a => Eq (BST a) where
    Empty == Empty = True
    Node info1 l1 r1 == Node info2 l2 r2 = info1 == info2 && l1 == l2 && r1 == r2
    _ == _ = False
```

H. a verifică egalitatea dintre doi (BST a), și nuie să putem compara informația din noduri.
=> Afhul(BST a) e o instanță a clasei Eq dacă 'a' e instanță a clasei Eq (= constrangere)

EXTINDERE DE CLASE

```
class (Eq a) => Ord a where
    compare          :: a -> a -> Ordering
    ((<), (<=), (>) , (>=)) :: a -> a -> Bool
    max, min         :: a -> a -> a

    compare x y = if x == y then EQ
                   else if x <= y then LT
                   else GT

    x < y = case compare x y of { LT -> True; _ -> False }
    x <= y = case compare x y of { GT -> False; _ -> True }
    x > y = case compare x y of { GT -> True; _ -> False }
    x >= y = case compare x y of { LT -> False; _ -> True }

    max x y = if x <= y then y else x
    min x y = if x <= y then x else y
```

- un tip 'a' poate face parte din Ord doar dacă e instantiată.
- adică: pentru a ordona elemente trebuie să putem defini egalitatea dintre ele.

CLASE PREDEFINITE

Ex: Eq, Ord, Show, ...

DERIVING

→ keyword prin care vine genera
compilatorului să genereze o
implementare default a funcțiilor
unei clase predefinite