Lab9: Polimorfism + Clase

04 May 2023 20:08

TYPE VARIABLES

! OBS: Tipurile se scriu cu literă mare: Int, Bool, Char...

Ex: funcția head - primește o listă de orice tip

```
*Main> :t head
head :: [a] -> a
```

POLIMORFISM → PARAMETRIC
→ AD-HOC

1) POLIMORFISM PARAMETRIC

→ tipul unei valori conține una sau mai multe variabile de tip, dar nu și constrângeri

Ex:
- id :: a → a
- head :: [a] → a ← var de tip, ≠ tip
- length :: [a] → Int
- map :: (a → b) → [a] → [b]

} funcții polimorfice

⇒ ac. implementare indif. de tip

! În contexte diferite, a ia tipuri diferite

head [1, 2, 3] → head :: [Int] → Int
head ['a', 'b', 'c'] → head :: [Char] → Char

! Dc. o var. de tip apare de mai multe ori, se înlocuiește cu ac. tip peste tot:

zipWith (\x y → (x, y)) [1,2,3] ['a', 'b', 'c']

zipWith :: a → b → c → [a] → [b] → [c]

zipWith :: a → b → c → [a] → [b] → [c]

⇒ a = Int, b = Char, c = [(a, a)] - [(Int, Char)]

② POLIMORFISM AD-HOC

→ implementări diferite pentru tipuri diferite + constrângeri de tip

(=) "SUPRASCRIEREA"

TYPECLASES + INSTANȚIERE

→ inspirate din Java - colecții de funcții

Eq → interfață/clasă pt. verif. egalității

```
class Eq a where
  (==), (/=) :: a → a → Bool
  (==) x y = not ((/=) x y)
  (/=) x y = not ((==) x y)
```

membri implicați

implementări din interfață

INSTANȚIERE EQ!

```
instance Eq Int where
```

[instance Eq t/p where
...
] SINTAXA

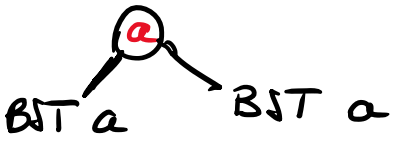
- data Person = Person { name :: String, age :: Int }

```
instance Eq Person where
  (==) person1 person2 = (name person1
       == name person2) &&
       (age person1 == age person2)
```

p1 = Person ...
p2 = Person ...
p1 == p2

- data BST a = EmptyNode | Node a (BST a) (BST a)

0 - EMPTY



:: a
- Node info left right

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

=> (BST a) e in Eq <=> a este instanta Eq

EXTINDERE DE CLASE

Tb. să instanțieze Eq

```
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
```

---

6. Supraîncărcați în Haskell operatorul de egalitate pentru funcții unare cu parametru numeric, astfel încât două funcții să fie considerate egale dacă valorile lor coincid în cel puțin 10 puncte din intervalul $1, \ldots, 100$.

$a \to b$

clasa Eq

```
instance (Num a, Eq b) => Eq (a -> b) where
  f == g = length (filter (\x -> f x == g x) [1..100]) >= 10
```