

Paradigme de Programare

Conf. dr. ing. Andrei Olaru

andrei.olaru@upb.ro

Departamentul de Calculatoare

2024

Cursul 7: Tipuri în Haskell

```
> :t map
```

```
map :: (a -> b) -> [a] -> [b]
```

```
> map ++ 5 tail [1..]
```

```
Couldn't match expected type '[a0]'
```

```
with actual type '(a1 -> b0) -> [a1] -> [b0]'
```

```
In the first argument of '(++)', namely 'map'
```

```
In the expression: map ++ 5 tail [1 .. ]
```

```
In an equation for 'it': it = map ++ 5 tail [1 .. ]
```



1 Tipare

2 Sinteză de tip

3 TDA

Tipare

- Tipuri ca **mulțimi** de valori:
 - Bool = {True, False}
 - Natural = {0, 1, 2, ...}
 - Char = {'a', 'b', 'c', ...}
- **Rolul** tipurilor (vezi cursuri anterioare);
- Tipare **statică**:
 - etapa de tipare **anterioară** etapei de evaluare;
 - asocierea **fiecărei** expresii din program cu un tip;
- Tipare **tare**: absența conversiilor **implicite** de tip;
- Expresii de:
 - **program**: 5, 2 + 3, x && (not y)
 - **tip**: Integer, [Char], Char -> Bool, a



Exemplu

```
1 5 :: Integer
2 'a' :: Char
3 (+1) :: Integer -> Integer
4 [1,2,3] :: [Integer] -- liste de un singur tip !
5 (True, "Hello") :: (Bool, [Char])
6 etc.
```

- Tipurile de bază sunt tipurile elementare din limbaj:

Bool, Char, Integer, Int, Float, ...

- Reprezentare uniformă:

```
1 data Integer = ... | -2 | -1 | 0 | 1 | 2 | ...
2 data Char = 'a' | 'b' | 'c' | ...
```

⇒ tipuri noi pentru valori sau funcții

- **Funcții** de tip, ce îmbogățesc tipurile din limbaj.

Ex Constructori de tip predefiniți

```
1  -- Constructorul de tip functie: ->
2  (-> Bool Bool) ⇒ Bool -> Bool
3  (-> Bool (Bool -> Bool)) ⇒ Bool -> (Bool -> Bool)
4
5  -- Constructorul de tip lista: []
6  ([] Bool) ⇒ [Bool]
7  ([] [Bool]) ⇒ [[Bool]]
8
9  -- Constructorul de tip tuplu: (, ..., )
10 ((,) Bool Char) ⇒ (Bool, Char)
11 ((,,) Bool ((,) Char [Bool]) Bool)
12           ⇒ (Bool, (Char, [Bool]), Bool)
```



- Constructorul `->` este asociativ **dreapta**:

`Integer -> Integer -> Integer`

\equiv `Integer -> (Integer -> Integer)`

Ex Exemplu

```
1 add6      :: Integer -> Integer -> Integer
2 add6 x y  =   x + y
3
4 f         :: (Integer -> Integer) -> Integer
5 f g      =   (g 3) + 1
6
7 idd      :: a -> a           -- functie polimorfica
8 idd x    =   x             -- a: variabila de tip!
```


Sinteză de tip



+ **Sinteză de tip – *type inference*** – Determinarea **automată** a tipului unei expresii, pe baza unor reguli precise.

- Adnotările **explicite** de tip, deși posibile, **necesare** în majoritatea cazurilor
- Dependentă de:
 - **componentele** expresiei
 - **contextul** lexical al expresiei
- Reprezentarea tipurilor → **expresii** de tip:
 - **constante** de tip: tipuri de bază;
 - **variabile** de tip: pot fi legate la orice expresii de tip;
 - **aplicații** ale constructorilor de tip pe expresii de tip.



+ **Progres** O expresie bine-tipată (căreia i se poate asocia un tip):

- este o **valoare** (nu este o aplicare de funcție) *sau*
- (este aplicarea unei funcții și) poate fi **redușă** (vezi β -redex).

+ **Conservare** Evaluarea unei expresii bine-tipate produce o expresie **bine-tipată** – de obicei, cu același tip.

- dacă **sinteza de tip** pentru expresia E dă tipul t , atunci după reducere, valoarea expresiei E va fi de tipul t .

Exemple de sinteză de tip

Câteva reguli simplificate de sinteză de tip

- Formă:
$$\frac{\text{premise-1} \dots \text{premise-m}}{\text{concluzie-1} \dots \text{concluzie-n}} \text{ (nume)}$$

- Funcție:
$$\frac{\text{Var} :: a \quad \text{Expr} :: b}{\backslash \text{Var} \rightarrow \text{Expr} :: a \rightarrow b} \text{ (TLambda)}$$
- Aplicație:
$$\frac{\text{Expr1} :: a \rightarrow b \quad \text{Expr2} :: a}{(\text{Expr1 Expr2}) :: b} \text{ (TApp)}$$
- Operatorul +:
$$\frac{\text{Expr1} :: \text{Int} \quad \text{Expr2} :: \text{Int}}{\text{Expr1} + \text{Expr2} :: \text{Int}} \text{ (T+)}$$
- Literalii întregi:
$$\frac{}{0, 1, 2, \dots :: \text{Int}} \text{ (TInt)}$$



Ex Exemplul 1

$$1 \quad f \circ g = (g \circ 3) + 1$$



Ex Exemplul 1

1 $f \ g = (g \ 3) + 1$

$$\frac{g :: a \quad (g \ 3) + 1 :: b}{f :: a \rightarrow b} \quad (\text{TLambda})$$

$$\frac{(g \ 3) :: \text{Int} \quad 1 :: \text{Int}}{(g \ 3) + 1 :: \text{Int}} \quad (\text{T+})$$

$$\Rightarrow b = \text{Int}$$

$$\frac{g :: c \rightarrow d \quad 3 :: c}{(g \ 3) :: d} \quad (\text{TApp})$$

$$\Rightarrow a = c \rightarrow d, \quad c = \text{Int}, \quad d = \text{Int}$$

$$\Rightarrow f :: (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$$



Ex Exemplul 2

```
1 fix f = f (fix f)
```



Ex Exemplul 2

1 `fix f = f (fix f)`

$$\frac{f :: a \quad f \text{ (fix f) } :: b}{\text{fix} :: a \rightarrow b} \text{ (TLambda)}$$

$$\frac{f :: c \rightarrow d \quad (\text{fix f}) :: c}{(f \text{ (fix f)}) :: d} \text{ (TApp)}$$

$$\Rightarrow a = c \rightarrow d, b = d$$

$$\frac{\text{fix} :: e \rightarrow g \quad f :: e}{(\text{fix f}) :: g} \text{ (TApp)}$$

$$\Rightarrow a \rightarrow b = e \rightarrow g, a = e, b = g, c = g$$

$$\Rightarrow \text{fix} :: (c \rightarrow d) \rightarrow b = (g \rightarrow g) \rightarrow g$$



Ex Exemplul 3

$$1 \quad f(x) = (x \ x)$$



Ex Exemplul 3

1 $f\ x = (x\ x)$

$$\frac{x :: a \quad (x\ x) :: b}{f :: a \rightarrow b} \quad (\text{TLambda})$$

$$\frac{x :: c \rightarrow d \quad x :: c}{(x\ x) :: d} \quad (\text{TApp})$$

Ecuția $c \rightarrow d = c$ **nu** are soluție (\nexists tipuri recursive)
 \Rightarrow funcția **nu** poate fi tipată.



- la baza sintezei de tip: **unificarea** → legarea variabilelor în timpul procesului de sinteză, în scopul **unificării** diverselor formule de tip elaborate.

+ **Unificare** Procesul de identificare a valorilor **variabilelor** din 2 sau mai multe formule, astfel încât **substituirea** variabilelor prin valorile asociate să conducă la **coincidența** formulelor.

+ **Substituție** O substituție este o mulțime de **legări** variabilă - valoare.



- O **variabilă de tip** a unifică cu o **expresie de tip** E doar dacă:
 - $E = a$ *sau*
 - $E \neq a$ și E nu conține a (*occurrence check*).
Exemplu: a unifică cu $b \rightarrow c$ dar nu cu $a \rightarrow b$.
- **2 constante** de tip unifică doar dacă sunt egale;
- **2 aplicații** de tip unifică doar dacă implică același constructor de tip și argumente ce unifică recursiv.



Ex Exemplu

- Pentru a unifica expresiile de tip:
 - $t1 = (a, [b])$
 - $t2 = (\text{Int}, c)$



Exemplu

- Pentru a unifica expresiile de tip:
 - $t1 = (a, [b])$
 - $t2 = (Int, c)$
- putem avea substituțiile (variante):
 - $S1 = \{a \leftarrow Int, b \leftarrow Int, c \leftarrow [Int]\}$
 - $S2 = \{a \leftarrow Int, c \leftarrow [b]\}$
- Forme comune pentru $s1$ respectiv $s2$:
 - $t1/S1 = t2/S1 = (Int, [Int])$
 - $t1/S2 = t2/S2 = (Int, [b])$

+ **Most general unifier – MGU** Cea mai **generală** substituție sub care formulele unifică. Exemplu: $s2$.



Ex Exemplu

- Tipurile: $t1 = (a, [b])$, $t2 = (\text{Int}, c)$
 - MGU: $S = \{a \leftarrow \text{Int}, c \leftarrow [b]\}$
 - Tipuri mai particulare (instanțe): $(\text{Integer}, [\text{Integer}])$, $(\text{Integer}, [\text{Char}])$, etc
- Funcția: $\lambda x . x$
 - Tipuri corecte: $\text{Int} \rightarrow \text{Int}$, $\text{Bool} \rightarrow \text{Bool}$, $a \rightarrow a$

+ **Tip principal al unei expresii** – Cel mai **general** tip care descrie **complet** natura expresiei. Se obține prin utilizarea MGU.

TDA

Constructorul de tip Natural

Exemplu de definire TDA 1



Ex Exemplu

```
1 data Natural      = Zero
2                   | Succ Natural
3   deriving (Show, Eq)
4
5 unu                = Succ Zero
6 doi                = Succ unu
7
8 addNat Zero n      = n
9 addNat (Succ m) n  = Succ (addNat m n)
```



- Constructor de **tip**: Natural
 - nular;
 - **se confundă** cu tipul pe care-l construiește.
- Constructori de **date**:
 - Zero: nular
 - Succ: unar
- Constructorii de date ca **funcții**, dar utilizabile în *pattern matching*.

```
1 Zero :: Natural
2 Succ :: Natural -> Natural
```

Constructorul de tip `Pair`

Exemplu de definire TDA 2



Ex Exemplu

```
1 data Pair a b    = P a b
2   deriving (Show, Eq)
3
4 pair1            = P 2 True
5 pair2            = P 1 pair1
6
7 myFst (P x y)    = x
8 mySnd (P x y)    = y
```



- Constructor de `tip`: `Pair`
 - polimorfic, binar;
 - generează un tip în momentul **aplicării** asupra 2 tipuri.

- Constructor de `date`: `P`, binar:

```
1 P :: a -> b -> Pair a b
```



- tipuri în Haskell
- expresii de tip și construcție de tipuri
- sinteză de tip, unificare

+ Dați feedback la acest curs aici:

[<https://docs.google.com/forms/d/e/1FAIpQLSeY7VuAt5n6hyHHnNUplLWfWt7UkJBGhkviewform>]

