

# PARADIGME DE PROGRAMARE

Curs 7b  
Sinteză de tip.

1

## Sinteză de tip – Cuprins

- Sistem de tipuri
- Sinteză de tip
- Siguranță de tip
- Unificare
- Exemple
- Mesaje de eroare

2

2

## Sistem de tipuri

**Sistem de tipuri** = set de mecanisme și reguli valabile într-un limbaj de programare, privind organizarea, construirea, manevrarea tipurilor de date acceptate în limbaj

**Atribuții** (nu se ocupă neapărat de toate)

- Asocierea de tipuri constructelor din limbaj
- Definirea de noi tipuri
- Decizia asupra echivalenței / compatibilității tipurilor
- Sinteza (inferența) de tip

3

3

## Avantaje și limitări

**Avantaje**

- Detectare timpurie a
  - Erorilor (+ mai mare acuratețe în identificarea sursei)
  - Codului inutil sau invalid
- Declarații de tip – abstractizare, documentare (care rămâne consistentă cu modificările efectuate ulterior)
- Eficiență (se pot optimiza operații pentru anumite tipuri)

**Limitări**

- Rejectează cod care nu ar produce probleme
- Renunță la anumite facilități (ex: Haskell nu permite liste eterogene)
- Tiparea statică necesită ori adnotări (declarații) de tip, ori sinteză (inferență) de tip

4

4

## Sinteză de tip – Cuprins

- Sistem de tipuri
- Sinteză de tip
- Siguranță de tip
- Unificare
- Exemple
- Mesaje de eroare

5

5

## Sinteză (inferență) de tip

- Tipul unei expresii este sintetizat în funcție de
  - **Tipul componentelor** expresiei (ex: tipul expresiei  $z + 2$ )
  - **Contextul** lexical al expresiei (ex: tipul lui  $x$  în contextul  $[ 'a' .. x ]$ )
- Reprezentarea tipului – expresie de tip, care reprezintă o combinație de
  - **Constante de tip** (tipurile primitive: Bool, Char, Int, etc.)
  - **Variabile de tip** (cu semnificația „orice (expresie de) tip”)
  - **Constructorii de tip** ( $(, [ , ->$ , constructori definiți cu „data”)

**Sinteză de tip** = determinarea automată a tipului unei expresii, prin combinarea constantelor, variabilelor și constructorilor de tip după reguli bine stabilite (reguli de sinteză de tip)

6

6

## Reguli de sinteză de tip

### Relație de tipare: $e :: T$

- Se citește „expresia  $e$  are tipul  $T$ ”
- O expresie  $e$  este **bine tipată**  $\Leftrightarrow$  există un tip  $T$  astfel încât  $e :: T$

### Regulă de sinteză de tip

$P_1, P_2, \dots, P_n$

$\frac{}{C}$  (id\_regulă)

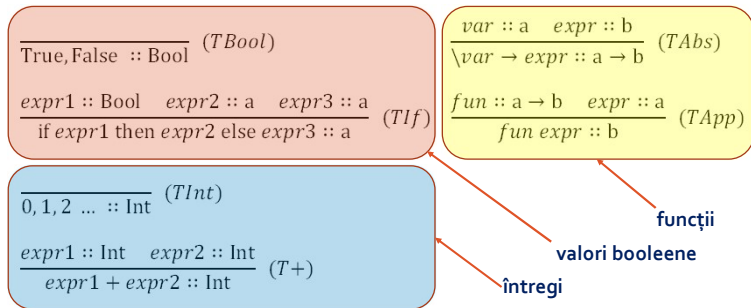
$P_i$  = premise (0 sau mai multe relații de tipare)

$C$  = concluzie (o relație de tipare)

7

7

## Reguli de sinteză de tip – Exemple



8

8

## Observații

- Tipul unei expresii este **unic**
- Regulile de tipare funcționează în ambele sensuri (vom observa că în sinteza de tip pornim de la concluzie către premise)
  - De la **premise către concluzie**  
**Exemplu:** Dacă  $expr1 :: Int$  și  $expr2 :: Int$ , atunci  $expr1 + expr2 :: Int$
  - De la concluzie către premise  
**Exemplu:** Dacă  $expr1 + expr2 :: Int$ , atunci  $expr1 :: Int$  și  $expr2 :: Int$

9

9

## Algoritm de sinteză de tip

- **Identifică regula aplicabilă**, prin pattern match între
  - Expresia care trebuie tipată
  - Concluziile regulilor de sinteză de tip
- **Aplică regula**, obținând constrângerile asupra componentelor expresiei
- **Pentru fiecare componentă în parte, reia algoritmul**, până când nu mai este nimic de descompus în părți componente
- **Tipul expresiei rezultă din aplicarea tuturor constrângerilor** obținute la pașii anteriori

10

10

## Sinteză de tip – Exemple

Care este tipul funcției  $f$ :  $f\ g = g\ 3 + 1 \Leftrightarrow f = \lambda g \rightarrow g\ 3 + 1$

Constrângeri

$$\frac{var :: a \quad expr :: b}{\lambda var \rightarrow expr :: a \rightarrow b} (TAbs) \quad \frac{g :: a \quad g\ 3 + 1 :: b}{f :: a \rightarrow b} (TAbs)$$

$$\frac{expr1 :: Int \quad expr2 :: Int}{expr1 + expr2 :: Int} (T+) \quad \frac{g\ 3 :: Int \quad 1 :: Int}{g\ 3 + 1 :: Int} (T+)$$

$$\frac{fun :: a \rightarrow b \quad expr :: a}{fun\ expr :: b} (TApp) \quad \frac{g :: c \rightarrow Int \quad 3 :: c}{g\ 3 :: Int} (TApp)$$

$$\frac{}{0, 1, 2 \dots :: Int} (TInt) \quad \frac{}{1, 3 :: Int} (TInt)$$

Așadar  $f :: a \rightarrow b \Leftrightarrow f :: (c \rightarrow Int) \rightarrow Int \Leftrightarrow f :: (Int \rightarrow Int) \rightarrow Int$

$b = Int$

$a = c \rightarrow Int$

$c = Int$

11

11

## Sinteză de tip – Exemple

Care este tipul funcției  $fix$ :  $fix\ f = f\ (fix\ f) \Leftrightarrow fix = \lambda f \rightarrow f\ (fix\ f)$

Constrângeri

$$\frac{var :: a \quad expr :: b}{\lambda var \rightarrow expr :: a \rightarrow b} (TAbs) \quad \frac{f :: a \rightarrow f\ (fix\ f) :: b}{fix :: a \rightarrow b} (TAbs)$$

$$\frac{fun :: a \rightarrow b \quad expr :: a}{fun\ expr :: b} (TApp) \quad \frac{f :: c \rightarrow b \quad (fix\ f) :: c}{f\ (fix\ f) :: b} (TApp)$$

$$\frac{fun :: a \rightarrow b \quad expr :: a}{fun\ expr :: b} (TApp) \quad \frac{fix :: d \rightarrow c \quad f :: d}{fix\ f :: c} (TApp)$$

Așadar  $fix :: a \rightarrow b \Leftrightarrow fix :: (c \rightarrow b) \rightarrow b \Leftrightarrow fix :: (b \rightarrow b) \rightarrow b$

$a = c \rightarrow b$

$a = d, b = c$

12

12

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru expresia din stânga este corectă semnătura din dreapta?

<code>(++)</code>	<code>[a] -&gt; [b] -&gt; [c]</code>
<code>zipWith</code> (ex: <code>zipWith (+) [1,2] [3,4] = [4,6]</code> )	<code>(a -&gt; a -&gt; a) -&gt; [a] -&gt; [a] -&gt; [a]</code>
<code>unzip</code> (ex: <code>unzip [(1,2), (3,4)] = [(1,3), [2,4]]</code> )	<code>[(a,b)] -&gt; ([a], [b])</code>
<code>[take, drop]</code>	<code>[Int -&gt; [a] -&gt; [a]]</code>
<code>words</code> (ex: <code>words "wait for it" = ["wait", "for", "it"]</code> )	<code>String -&gt; String</code>
<code>(odd . fst)</code>	<code>(Int, b) -&gt; Bool</code>
<code>map ("o1" :) . reverse</code>	<code>[[Char]] -&gt; [[Char]]</code>

13

13

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru expresia din stânga este corectă semnătura din dreapta?

<code>(++)</code>	<code>[a] -&gt; [b] -&gt; [c]</code>
<code>zipWith</code> (ex: <code>zipWith (+) [1,2] [3,4] = [4,6]</code> )	<code>(a -&gt; a -&gt; a) -&gt; [a] -&gt; [a] -&gt; [a]</code>

Nu, nu pot concatena liste cu elemente de tipuri diferite!

**Ex:** `[1,2,3] ++ [(+),(-)]`      **Corect:** `[a] -> [a] -> [a]`

14

14

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru expresia din stânga este corectă semnătura din dreapta?

<code>zipWith</code> (ex: <code>zipWith (+) [1,2] [3,4] = [4,6]</code> )	<code>(a -&gt; a -&gt; a) -&gt; [a] -&gt; [a] -&gt; [a]</code>
<code>unzip</code> (ex: <code>unzip [(1,2), (3,4)] = [(1,3), [2,4]]</code> )	<code>[(a,b)] -&gt; ([a], [b])</code>

Nu, nu e necesar ca cele două liste să aibă elemente de același tip.

**Ex:** `zipWith (:) [1,2] [[3],[4]]`      **Corect:** `(a -> b -> c) -> [a] -> [b] -> [c]`

15

15

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru expresia din stânga este corectă semnătura din dreapta?

<code>unzip</code> (ex: <code>unzip [(1,2), (3,4)] = [(1,3), [2,4]]</code> )	<code>[(a,b)] -&gt; ([a], [b])</code>
<code>[take, drop]</code>	<code>[Int -&gt; [a] -&gt; [a]]</code>

**Ex:** `unzip [(1,'a'), (2,'b'), (3,'c')]`

16

16

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru **expresia** din stânga este corectă semnătura din dreapta?

`[take, drop]` `[[Int -> [a] -> [a]]`  
`words` (ex: `words "wait for it" = ["wait", "for", "it"]`) `String -> String`

Da, atenție la faptul că semnătura indică o **listă de funcții de același tip**.

17

17

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru **expresia** din stânga este corectă semnătura din dreapta?

Nu, rezultatul este o listă de String-uri, nu o listă de caractere (String ~ [Char]).

**Corect:** `String -> [String]`

Alternativ, se putea scrie: `words :: [Char] -> [[Char]]`

`words` (ex: `words "wait for it" = ["wait", "for", "it"]`) `String -> String`  
`(odd . fst)` `(Int, b) -> Bool`

18

18

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru **expresia** din stânga este corectă semnătura din dreapta?

Da:

`fst :: (a,b) -> a` `odd :: Integral a => a -> Bool`

**Ex:** `(odd . fst) (1, [])`

`(odd . fst)` `(Int, b) -> Bool`

`map ("o1":) . reverse` `[[Char]] -> [[Char]]`

19

19

## Antrenament: Expresii și semnături

**Exercițiu:** Pentru **expresia** din stânga este corectă semnătura din dreapta?

Nu, fiecare element din listă trebuie să fie nu un String, ci o listă de String-uri.

"o1" : element înseamnă că "o1" are tipul a și element are tipul [a].

**Ex:** `(map ("o1":) . reverse) ["So", "many", ["Strings"]]`

**Corect:** `[[String]] -> [[String]]` sau `[[Char]] -> [[Char]]`

`map ("o1":) . reverse` `[[Char]] -> [[Char]]`

20

20

## Sinteză de tip – Cuprins

- Sistem de tipuri
- Sinteză de tip
- Siguranță de tip
- Unificare
- Exemple
- Mesaje de eroare

21

21

## Siguranță de tip

Există **siguranță de tip** atunci când expresiile bine tipate „nu o pot lua pe căi greșite” în timpul evaluării, adică atunci când se respectă proprietățile:

- **Progres**
  - O expresie bine tipată  $e :: T$  nu se blochează:
    - Ori este o valoare
    - Ori se poate reduce conform unei reguli de evaluare
- **Conservare**
  - Un pas de evaluare asupra unei expresii bine tipate  $e :: T$  produce o expresie bine tipată  $e' :: T$

Scop: prevenirea erorilor cauzate de discrepanțele dintre tipurile așteptate și cele procesate.

22

22

## Test

Scrieți un list comprehension în Haskell care reprezintă mulțimea (așadar fără duplicate) tuturor tripletelor pitagoreice de numere naturale de maxim 3 cifre.

**Obs:** Numerele  $a, b, c$  formează un triplet pitagoreic dacă pot fi laturile unui triunghi dreptunghic ( $a^2 + b^2 = c^2$ ).

**Ex:** 3, 4, 5 formează un triplet pitagoreic întrucât  $9 + 16 = 25$

23

23

## Sinteză de tip – Cuprins

- Sistem de tipuri
- Sinteză de tip
- Siguranță de tip
- Unificare
- Exemple
- Mesaje de eroare

24

24

## Unificare

Unificarea a 2 expresii de tip = găsirea celei mai generale substituții

$$S = \{e_1 / t_1, e_2 / t_2, \dots, e_n / t_n\}$$

pentru variabilele de tip  $t_i$  din cele 2 expresii astfel încât, în urma substituției, cele 2 expresii de tip devin una și aceeași

- Se folosește în sinteza de tip în pasul de aplicare a constrângerilor

### Exemplu

- Din sinteza de tip avem constrângerile  $a = b \rightarrow [d]$  și  $a = \text{Int} \rightarrow c$   
 $\rightarrow b \rightarrow [d]$  trebuie să unifice cu  $\text{Int} \rightarrow c$ , altfel sinteza de tip va eșua
- Există  $S = \{\text{Int} / b, [d] / c\}$  a.f.  $b \rightarrow [d]$  și  $\text{Int} \rightarrow c$  se intersectează în  $\text{Int} \rightarrow [d]$
- Există și  $S' = \{\text{Int} / b, [\text{Int}] / c, \text{Int} / d\}$ , rezultând în  $\text{Int} \rightarrow [\text{Int}]$ , dar  $S$  este mai generală!

25

25

## Mai multe exemple

- $e_1 = a, e_2 = \text{Int} \rightarrow \text{Bool}$
- $e_1 = (a, [a]), e_2 = ([b], c)$
- $e_1 = \text{Int}, e_2 = \text{Float}$

26

26

## Mai multe exemple

- $e_1 = a, e_2 = \text{Int} \rightarrow \text{Bool}$        $S = \{\text{Int} \rightarrow \text{Bool} / a\}$        $e_{\text{unificată}} = \text{Int} \rightarrow \text{Bool}$



- $e_1 = (a, [a]), e_2 = ([b], c)$

- $e_1 = \text{Int}, e_2 = \text{Float}$

27

## Mai multe exemple

- $e_1 = a, e_2 = \text{Int} \rightarrow \text{Bool}$        $S = \{\text{Int} \rightarrow \text{Bool} / a\}$        $e_{\text{unificată}} = \text{Int} \rightarrow \text{Bool}$



- $e_1 = (a, [a]), e_2 = ([b], c)$        $S = \{[b] / a, [[b]] / c\}$        $e_{\text{unificată}} = ([b], [[b]])$



- $e_1 = \text{Int}, e_2 = \text{Float}$

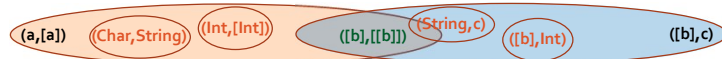
28

## Mai multe exemple

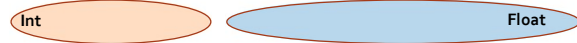
- $e_1 = a, e_2 = \text{Int} \rightarrow \text{Bool}$       $S = \{\text{Int} \rightarrow \text{Bool} / a\}$       $e_{\text{unificată}} = \text{Int} \rightarrow \text{Bool}$



- $e_1 = (a, [a]), e_2 = ([b], c)$       $S = \{[b] / a, [[b]] / c\}$       $e_{\text{unificată}} = ([b], [[b]])$



- $e_1 = \text{Int}, e_2 = \text{Float}$      Nu unifică (nu se fac conversii implicite)!



29

29

## Unificare la sinteza de tip – Exemplu

Care este tipul funcției  $f$ :  $f x = x x \Leftrightarrow f = \lambda x \rightarrow x x$

Constrângeri

$$\frac{\text{var} :: a \quad \text{expr} :: b}{\backslash \text{var} \rightarrow \text{expr} :: a \rightarrow b} \quad (TAbs) \qquad \frac{x :: a \quad x x :: b}{f :: a \rightarrow b} \quad (TAbs)$$

$$\frac{\text{fun} :: a \rightarrow b \quad \text{expr} :: a}{\text{fun expr} :: b} \quad (TApp) \qquad \frac{x :: c \rightarrow b \quad x :: c}{x x :: b} \quad (TApp)$$

$a = c \rightarrow b = c$

$a$  unifică cu  $c \rightarrow b$ , dar  $c \rightarrow b$  nu unifică cu  $c$ .

$(c = c \rightarrow b = c \rightarrow b \rightarrow b = c \rightarrow b \rightarrow b \rightarrow b \dots)$  rezultă un tip infinit, care nu poate fi sintetizat)

Întrucât unificarea eșuează (găsește o inconsistență), definiția lui  $f$  generează eroare:

Occurs check: cannot construct the infinite type:  $t_1 \sim t_1 \rightarrow t$

30

30

## Reguli folosite în algoritmul de unificare

- O **variabilă de tip**  $a$  unifică cu o **expresie de tip**  $e$  ( $S = \{e / a\}$ ) dacă și numai dacă
  - $e = a$  sau
  - $e$  nu conține  $a$  (occurs check – pentru a evita tipuri infinite)
- **2 constante de tip** unifică ( $S = \{\}$ ) dacă și numai dacă sunt egale
- **2 construcții de tip** unifică ( $S = S_1 \cup \dots \cup S_n$ ) dacă și numai dacă sunt aplicații ale aceluiași constructor de tip asupra câte  $n$  argumente care unifică recursiv (prin substituțiile  $S_1 \dots S_n$ )

31

31

## Unificare la sinteza de tip – Implementare

- Algoritmul de sinteză de tip pe bază de unificare folosește
  - O **stivă de constrângeri** de forma  $e_1 = e_2$  – se umple în timpul algoritmului de sinteză de tip
  - O **substituție S** (mulțime de asocieri între variabile de tip și expresii de tip) – inițial vidă
- La fiecare iterație a pasului de unificare
  - Se extrage o constrângere  $e_1 = e_2$  din stivă
  - Dacă  $e_1$  unifică cu  $e_2$  prin substituția  $S_{12}$ , se aplică  $S_{12}$ 
    - asupra stivei și
    - asupra substituției S
  - Altfel, unificarea eșuează  $\rightarrow$  sinteza de tip eșuează  $\rightarrow$  eroare de tip

32

32



## Exemplu

Stiva	Substituția
a = b -> c a = d -> d b = Int	
b -> c = d -> d b = Int	a = b -> c

33

33

## Exemplu

Stiva	Substituția
a = b -> c a = d -> d b = Int	
b -> c = d -> d b = Int	a = b -> c
b = d c = d b = Int	a = b -> c

34

34

## Exemplu

Stiva	Substituția
a = b -> c a = d -> d b = Int	
b -> c = d -> d b = Int	a = b -> c
b = d c = d b = Int	a = b -> c
c = d d = Int	a = d -> c b = d

35

35

## Exemplu

Stiva	Substituția
a = b -> c a = d -> d b = Int	
b -> c = d -> d b = Int	a = b -> c
b = d c = d b = Int	a = b -> c
c = d d = Int	a = d -> c b = d
d = Int	a = d -> d b = d c = d

36

36

## Exemplu

Stiva	Substituția
a = b -> c a = d -> d b = Int	
b -> c = d -> d b = Int	a = b -> c
b = d c = d b = Int	a = b -> c
c = d d = Int	a = d -> c b = d
d = Int	a = d -> d b = d c = d
	a = Int -> Int b = Int c = Int d = Int

37

37

## Sinteză de tip – Cuprins

- Sistem de tipuri
- Sinteză de tip
- Siguranță de tip
- Unificare
- Exemple
- Mesaje de eroare

38

38

## Determinarea tipului expresiilor / funcțiilor

`length ([]++[1,2]) + length ([]++"abcd")`

Constrângeri

$(T+) \rightarrow \text{expr} :: \text{Int} \Leftrightarrow \text{length} ([]++[1,2]) :: \text{Int}, \text{length} ([]++\text{"abcd"}) :: \text{Int}$

$(TLen) \rightarrow []++[1,2] :: [a], []++\text{"abcd"} :: [b]$

$(T++) \rightarrow [] :: [a], [1,2] :: [a], [] :: [b], \text{"abcd"} :: [b]$

$(T[]) \rightarrow 1,2 :: a, 'a','b','c','d' :: b$

$(TInt, TChar) \rightarrow 1,2 :: \text{Int}, 'a','b','c','d' :: \text{Char}$

a = Int, b = Char

$[] :: [\text{Int}]$  și  $[] :: [\text{Char}]$  ok ( $[]$  e polimorfic – cele 2 liste vide nu reprezintă aceeași valoare)

Etapa de unificare reușește și ea (nu avem două constrângeri asupra aceleiași variabile), deci expresia are **tipul Int conform T+**.

39

39

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`

Constrângeri

40

40

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`      **Constrângeri**

`(T+) → expr :: Int ⇔ length (xs++[1,2]) :: Int, length (xs++"abcd") :: Int`

41

41

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`      **Constrângeri**

`(T+) → expr :: Int ⇔ length (xs++[1,2]) :: Int, length (xs++"abcd") :: Int`

`(TLen) → xs++[1,2] :: [a], xs++"abcd" :: [b]`

42

42

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`      **Constrângeri**

`(T+) → expr :: Int ⇔ length (xs++[1,2]) :: Int, length (xs++"abcd") :: Int`

`(TLen) → xs++[1,2] :: [a], xs++"abcd" :: [b]`

`(T++) → xs :: [a], [1,2] :: [a], xs :: [b], "abcd" :: [b]`      **[a] = [b]**

43

43

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`      **Constrângeri**

`(T+) → expr :: Int ⇔ length (xs++[1,2]) :: Int, length (xs++"abcd") :: Int`

`(TLen) → xs++[1,2] :: [a], xs++"abcd" :: [b]`

`(T++) → xs :: [a], [1,2] :: [a], xs :: [b], "abcd" :: [b]`      **[a] = [b]**

`(T[]) → 1,2 :: a, 'a','b','c','d' :: b`

44

44

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`      **Constrângeri**

`(T+) → expr :: Int ⇔ length (xs++[1,2]) :: Int, length (xs++"abcd") :: Int`

`(TLen) → xs++[1,2] :: [a], xs++"abcd" :: [b]`

`(T++) → xs :: [a], [1,2] :: [a], xs :: [b], "abcd" :: [b]`      **[a] = [b]**

`(T[]) → 1,2 :: a, 'a','b','c','d' :: b`

`(TInt,TChar) → 1,2 :: Int, 'a','b','c','d' :: Char`      **a = Int, b = Char**

45

45

## Determinarea tipului expresiilor / funcțiilor

`length (xs++[1,2]) + length (xs++"abcd")`      **Constrângeri**

`(T+) → expr :: Int ⇔ length (xs++[1,2]) :: Int, length (xs++"abcd") :: Int`

`(TLen) → xs++[1,2] :: [a], xs++"abcd" :: [b]`

`(T++) → xs :: [a], [1,2] :: [a], xs :: [b], "abcd" :: [b]`      **[a] = [b]**

`(T[]) → 1,2 :: a, 'a','b','c','d' :: b`

`(TInt,TChar) → 1,2 :: Int, 'a','b','c','d' :: Char`      **a = Int, b = Char**

În etapa de unificare: `[Int] = [Char] → Int = Char` care nu unifică, deci tipul expresiei nu poate fi sintetizat și vom obține o **eroare de tip**.

46

46

## Determinarea tipului expresiilor / funcțiilor

`curry id`      **Constrângeri**

47

47

## Determinarea tipului expresiilor / funcțiilor

`curry id`      **Constrângeri**

`(TApp) → expr :: b ⇔ curry :: a -> b, id :: a`

48

48

## Determinarea tipului expresiilor / funcțiilor

curry id

(TApp)  $\rightarrow$  expr :: b  $\Leftrightarrow$  curry :: a  $\rightarrow$  b, id :: a

$\frac{}{\text{curry} :: ((c, d) \rightarrow e) \rightarrow c \rightarrow d \rightarrow e}$  (TCurry)

Constrângeri

a = (c,d)  $\rightarrow$  e, b = c  $\rightarrow$  d  $\rightarrow$  e

49

49

## Determinarea tipului expresiilor / funcțiilor

curry id

(TApp)  $\rightarrow$  expr :: b  $\Leftrightarrow$  curry :: a  $\rightarrow$  b, id :: a

$\frac{}{\text{curry} :: ((c, d) \rightarrow e) \rightarrow c \rightarrow d \rightarrow e}$  (TCurry)

$\frac{}{\text{id} :: f \rightarrow f}$  (TId)

Constrângeri

a = (c,d)  $\rightarrow$  e, b = c  $\rightarrow$  d  $\rightarrow$  e

a = f  $\rightarrow$  f

50

50

## Determinarea tipului expresiilor / funcțiilor

curry id

(TApp)  $\rightarrow$  expr :: b  $\Leftrightarrow$  curry :: a  $\rightarrow$  b, id :: a

$\frac{}{\text{curry} :: ((c, d) \rightarrow e) \rightarrow c \rightarrow d \rightarrow e}$  (TCurry)

$\frac{}{\text{id} :: f \rightarrow f}$  (TId)

Constrângeri

a = (c,d)  $\rightarrow$  e, b = c  $\rightarrow$  d  $\rightarrow$  e

a = f  $\rightarrow$  f

În etapa de unificare: a = (c,d)  $\rightarrow$  e = f  $\rightarrow$  f  $\rightarrow$  e = f = (c,d)  $\rightarrow$  b = c  $\rightarrow$  d  $\rightarrow$  (c,d), deci expresia are tipul c  $\rightarrow$  d  $\rightarrow$  (c,d) conform TApp.

51

51

## Determinarea tipului expresiilor / funcțiilor

uncurry id

Constrângeri

52

52

## Determinarea tipului expresiilor / funcțiilor

uncurry id

$(TApp) \rightarrow expr :: b \Leftrightarrow uncurry :: a \rightarrow b, id :: a$

Constrângeri

53

53

## Determinarea tipului expresiilor / funcțiilor

uncurry id

$(TApp) \rightarrow expr :: b \Leftrightarrow uncurry :: a \rightarrow b, id :: a$

$\frac{}{uncurry :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e} (TUncurry)$

Constrângeri

$a = c \rightarrow d \rightarrow e, b = (c, d) \rightarrow e$

54

54

## Determinarea tipului expresiilor / funcțiilor

uncurry id

$(TApp) \rightarrow expr :: b \Leftrightarrow uncurry :: a \rightarrow b, id :: a$

$\frac{}{uncurry :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e} (TUncurry)$

$\frac{}{id :: f \rightarrow f} (TId)$

Constrângeri

$a = c \rightarrow d \rightarrow e, b = (c, d) \rightarrow e$

$a = f \rightarrow f$

55

55

## Determinarea tipului expresiilor / funcțiilor

uncurry id

$(TApp) \rightarrow expr :: b \Leftrightarrow uncurry :: a \rightarrow b, id :: a$

$\frac{}{uncurry :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e} (TUncurry)$

$\frac{}{id :: f \rightarrow f} (TId)$

Constrângeri

$a = c \rightarrow d \rightarrow e, b = (c, d) \rightarrow e$

$a = f \rightarrow f$

În etapa de unificare:  $a = c \rightarrow d \rightarrow e = f \rightarrow f \rightarrow c = f = d \rightarrow e \rightarrow b = (d \rightarrow e, d) \rightarrow e$ , deci expresia are tipul  $(d \rightarrow e, d) \rightarrow e$  conform TApp.

56

56

## Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

57

57

## Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

(TApp)  $\rightarrow$   $\text{expr} :: b \Leftrightarrow \text{curry} :: a \rightarrow b, \text{curry id} :: a$

58

58

## Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

(TApp)  $\rightarrow$   $\text{expr} :: b \Leftrightarrow \text{curry} :: a \rightarrow b, \text{curry id} :: a$

(ExAnterior)  $\rightarrow$   $\text{curry id} :: f \rightarrow g \rightarrow (f,g)$

$a = f \rightarrow g \rightarrow (f,g)$

59

59

## Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

(TApp)  $\rightarrow$   $\text{expr} :: b \Leftrightarrow \text{curry} :: a \rightarrow b, \text{curry id} :: a$

(ExAnterior)  $\rightarrow$   $\text{curry id} :: f \rightarrow g \rightarrow (f,g)$

$a = f \rightarrow g \rightarrow (f,g)$

$a = (c,d) \rightarrow e, b = c \rightarrow d \rightarrow e$

$\frac{}{\text{curry} :: ((c, d) \rightarrow e) \rightarrow c \rightarrow d \rightarrow e}$  (TCurry)

60

60

## Determinarea tipului expresiilor / funcțiilor

curry (curry id)

(TApp)  $\rightarrow$  expr :: b  $\Leftrightarrow$  curry :: a  $\rightarrow$  b, curry id :: a

(ExAnterior)  $\rightarrow$  curry id :: f  $\rightarrow$  g  $\rightarrow$  (f,g)

$\frac{}{\text{curry} :: ((c, d) \rightarrow e) \rightarrow c \rightarrow d \rightarrow e}$  (TCurry)

Constrângeri

a = f  $\rightarrow$  g  $\rightarrow$  (f,g)

a = (c,d)  $\rightarrow$  e, b = c  $\rightarrow$  d  $\rightarrow$  e

În etapa de unificare: a = f  $\rightarrow$  g  $\rightarrow$  (f,g) = (c,d)  $\rightarrow$  e  $\rightarrow$  f = (c,d), e = g  $\rightarrow$  (f,g) = g  $\rightarrow$  ((c,d),g)  
 $\rightarrow$  b = c  $\rightarrow$  d  $\rightarrow$  g  $\rightarrow$  ((c,d), g), deci expresia are tipul c  $\rightarrow$  d  $\rightarrow$  g  $\rightarrow$  ((c,d), g) conform TApp.

61

61

## Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

62

62

## Determinarea tipului expresiilor / funcțiilor

curry uncurry

(TApp)  $\rightarrow$  expr :: b  $\Leftrightarrow$  curry :: a  $\rightarrow$  b, uncurry :: a

Constrângeri

63

63

## Determinarea tipului expresiilor / funcțiilor

curry uncurry

(TApp)  $\rightarrow$  expr :: b  $\Leftrightarrow$  curry :: a  $\rightarrow$  b, uncurry :: a

$\frac{}{\text{uncurry} :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e}$  (TUncurry)

Constrângeri

a = (c  $\rightarrow$  d  $\rightarrow$  e)  $\rightarrow$  (c,d)  $\rightarrow$  e

64

64



## Determinarea tipului expresiilor / funcțiilor

### curry uncurry

$(TApp) \rightarrow expr :: b \Leftrightarrow curry :: a \rightarrow b, uncurry :: a$

$uncurry :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e$  (TUncurry)

$curry :: ((f, g) \rightarrow h) \rightarrow f \rightarrow g \rightarrow h$  (TCurry)

### Constrângeri

$a = (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e$

$a = (f, g) \rightarrow h, b = f \rightarrow g \rightarrow h$

65

65

## Determinarea tipului expresiilor / funcțiilor

### curry uncurry

$(TApp) \rightarrow expr :: b \Leftrightarrow curry :: a \rightarrow b, uncurry :: a$

$uncurry :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e$  (TUncurry)

$curry :: ((f, g) \rightarrow h) \rightarrow f \rightarrow g \rightarrow h$  (TCurry)

### Constrângeri

$a = (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e$

$a = (f, g) \rightarrow h, b = f \rightarrow g \rightarrow h$

În etapa de unificare:  $a = (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e = (f, g) \rightarrow h \rightarrow$  (printre altele  $c \rightarrow d \rightarrow e = (f, g)$  care nu unifică (sunt aplicații ale unor constructori de tip diferiți), deci tipul expresiei nu poate fi sintetizat și vom obține o eroare de tip.

**Exercițiu:** Încercați și `curry curry`, `uncurry curry`, `uncurry uncurry` (nu toate dau eroare).

66

66

## Sinteză de tip – Cuprins

- Sistem de tipuri
- Sinteză de tip
- Siguranță de tip
- Unificare
- Exemple
- Mesaje de eroare

67

67

## Parse error

- programul nu respectă **sintaxa Haskell**
- se indică locul în care a apărut un simbol neașteptat
- eroarea e la locul indicat sau undeva înainte

`f : Int -> [a] -> Int` curs7.hs:1:9: parse error on input '>'  
`f a b = a + length b`

`f x = x ++ [1; 2; 3]` curs7.hs:1:14: parse error on input ';'

`f x =` curs7.hs:2:9: parse error on input '|'  
 `| null x = 0`  
 `| otherwise = 1`

68

68

## Parse error

- programul nu respectă **sintaxa Haskell**
- se indică locul în care a apărut un simbol neașteptat
- eroarea e la locul indicat sau undeva înainte

```
f : Int -> [a] -> Int
f a b = a + length b
```

```
curs7.hs:1:9: parse error on input '->'
Înlocuiește cu ::
```

```
f x = x ++ [1; 2; 3]
```

```
curs7.hs:1:14: parse error on input ';'
Înlocuiește cu ,
```

```
f x =
  | null x = 0
  | otherwise = 1
```

```
curs7.hs:2:9: parse error on input '|'
Șterge = dinaintea de |
```

69

69

## Not in scope

- compilatorul **nu recunoaște o variabilă / o funcție / un constructor**
- fie am folosit o variabilă / o funcție / un constructor care nu există (posibil typo)
- fie trebuie să importăm modulul în care se află ace(a)sta

```
f a x
  | null x = cons a x
  | otherwise = f (a+1) (tail x)
```

Not in scope: `cons`

```
f a x
  | null x = a
  | oterwise = f (a+1) (tail x)
```

Not in scope: `oterwise`  
Perhaps you meant `otherwise`

```
f x = sort (take 10 x)
```

Not in scope: `sort`

70

70

## Not in scope

- compilatorul **nu recunoaște o variabilă / o funcție / un constructor**
- fie am folosit o variabilă / o funcție / un constructor care nu există (posibil typo)
- fie trebuie să importăm modulul în care se află ace(a)sta

```
f a x
  | null x = cons a x
  | otherwise = f (a+1) (tail x)
```

Not in scope: `cons`  
Înlocuiește cu (:)

```
f a x
  | null x = a
  | oterwise = f (a+1) (tail x)
```

Not in scope: `oterwise`  
Perhaps you meant `otherwise`

```
f x = sort (take 10 x)
```

Not in scope: `sort`

Caută funcția (ex: [hoogle.haskell.org](https://hoogle.haskell.org)) și vezi ce modul trebuie importat (aici `import Data.List`)

71

71

## The function <> is applied to <> arguments

- **am aplicat o funcție pe mai multe sau mai puține argumente decât așteaptă**
- fie am pus ceva în plus / minus
- fie n-am pus parantezele bine

```
import Data.List
f x = sort (take 10 x) (<)
```

The function `sort` is applied to two arguments,  
but its type `[a] -> [a]` has only one

```
f x = if null x then 0 else 1 + f tail x
```

...  
Probable cause: `tail` is applied to too few arguments

72

72

## The function <> is applied to <> arguments

- am aplicat o funcție pe mai multe sau mai puține argumente decât așteaptă
- fie am pus ceva în plus / minus
- fie n-am pus parantezele bine

```
import Data.List
f x = sort (take 10 x) (<)
f x = if null x then 0 else 1 + f tail x
```

The function 'sort' is applied to two arguments, but its type '[a]->[a]' has only one

Șterge (<)

...

Probable cause: 'tail' is applied to too few arguments

Pune paranteze la tail: f (tail x)

73

73

## Couldn't match expected type with actual type

- fie nu suntem în concordanță cu signatura (ori corectăm funcția, ori signatura)
- fie am aplicat o funcție pe tipuri incompatibile cu ce așteaptă funcția
  - posibil să fi greșit ordinea argumentelor

```
f :: String -> String
f = head

f = 'A' ++ "BC"

f = (:) "BC" 'A'
```

Couldn't match type 'Char' with '[Char]'

Couldn't match expected type '[Char]' with actual type 'Char'

Couldn't match expected type '[[Char]]' with actual type 'Char' In the second argument of '(:)', namely 'A'

74

74

## Couldn't match expected type with actual type

- fie nu suntem în concordanță cu signatura (ori corectăm funcția, ori signatura)
- fie am aplicat o funcție pe tipuri incompatibile cu ce așteaptă funcția
  - posibil să fi greșit ordinea argumentelor

```
f :: String -> String
f = head

f = 'A' ++ "BC"

f = (:) "BC" 'A'
```

Couldn't match type 'Char' with '[Char]'

Fie f :: String -> Char, fie f = (: []). head

Couldn't match expected type '[Char]' with actual type 'Char'

Fie 'A': "BC", fie "A" ++ "BC"

Couldn't match expected type '[[Char]]' with actual type 'Char' In the second argument of '(:)', namely 'A'

Schimb ordinea: (:) 'A' "BC"

75

75

## Non-exhaustive patterns

- o funcție definită cu pattern matching sau gărzi nu acoperă toate valorile tipului
- fie am uitat anumite valori
- fie avem un typo la numele funcției (și Haskell pur și simplu consideră că este altă funcție)

```
f x
  | x < 0 = -1
  | x > 0 = 1

mySum [] = 0
mySum (x:xs) = x + mySum xs
```

\*Main> f 0

\*\*\* Exception: curs7.hs:(1,1)-(3,19): Non-exhaustive patterns in function f (dacă avem warning-urile activate, primim warning la compilare)

\*Main> mySum [1..5]

\*\*\* Exception: ... Non-exhaustive patterns ... mySum

76

76

## Non-exhaustive patterns

- o funcție definită cu pattern matching sau gărzi **nu acoperă toate valorile tipului**
- fie am uitat anumite valori
- fie avem un typo la numele funcției (și Haskell pur și simplu consideră că este altă funcție)

```
f x
  | x < 0 = -1
  | x > 0 = 1

mySum [] = 0
mySum (x:xs) = x + mySum xs
```

\*Main> f 0  
\*\*\* Exception: curs7.hs:(1,1)-(3,19): Non-exhaustive patterns in function f  
(dacă avem warning-urile activate, primim warning la compilare)  
Adaugă | otherwise = ...

\*Main> mySum [1..5]  
\*\*\* Exception: ... Non-exhaustive patterns ... mySum  
Corectează mySUM în mySum

77

77

## Rezumat

Sinteză de tip

Regulă de sinteză de tip

Algoritm de sinteză de tip

Siguranță de tip

Unificare

O variabilă unifică cu

2 constante unifică

2 construcții unifică

Structuri folosite de algoritmul de unificare

78

78

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
Regulă de sinteză de tip

Algoritm de sinteză de tip

Siguranță de tip

Unificare

O variabilă unifică cu

2 constante unifică

2 construcții unifică

Structuri folosite de algoritmul de unificare

79

79

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip

**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)

Algoritm de sinteză de tip

Siguranță de tip

Unificare

O variabilă unifică cu

2 constante unifică

2 construcții unifică

Structuri folosite de algoritmul de unificare

80

80

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip**

Unificare  
 O variabilă unifică cu  
 2 constante unifică  
 2 construcții unifică

Structuri folosite de algoritmul de unificare

81

81

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip:** progres (expresia bine tipată nu se blochează), conservare (rămâne bine tipată după evaluare)  
 Unificare

O variabilă unifică cu  
 2 constante unifică  
 2 construcții unifică

Structuri folosite de algoritmul de unificare

82

82

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip:** progres (expresia bine tipată nu se blochează), conservare (rămâne bine tipată după evaluare)  
**Unificare:** cea mai generală substituție conform căreia 2 expresii de tip devin identice  
 O variabilă unifică cu  
 2 constante unifică  
 2 construcții unifică

Structuri folosite de algoritmul de unificare

83

83

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip:** progres (expresia bine tipată nu se blochează), conservare (rămâne bine tipată după evaluare)  
**Unificare:** cea mai generală substituție conform căreia 2 expresii de tip devin identice  
**O variabilă unifică cu:** orice, cât timp occurs check este OK  
 2 constante unifică  
 2 construcții unifică

Structuri folosite de algoritmul de unificare

84

84

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip:** progres (expresia bine tipată nu se blochează), conservare (rămâne bine tipată după evaluare)  
**Unificare:** cea mai generală substituție conform căreia 2 expresii de tip devin identice  
**O variabilă unifică cu:** orice, cât timp occurs check este OK  
**2 constante unifică:** dacă sunt identice  
**2 construcții unifică**

Structuri folosite de algoritmul de unificare

85

85

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip:** progres (expresia bine tipată nu se blochează), conservare (rămâne bine tipată după evaluare)  
**Unificare:** cea mai generală substituție conform căreia 2 expresii de tip devin identice  
**O variabilă unifică cu:** orice, cât timp occurs check este OK  
**2 constante unifică:** dacă sunt identice  
**2 construcții unifică:** dacă folosesc același constructor de tip pe argumente care unifică la rândul lor  
 Structuri folosite de algoritmul de unificare

86

86

## Rezumat

**Sinteză de tip:** determinarea tipului unei expresii pe baza regulilor de sinteză de tip  
**Regulă de sinteză de tip:** pe baza unor relații de tipare (premise) se obține o relație de tipare (concluzie)  
**Algoritm de sinteză de tip:** potrivește concluziile relațiilor de sinteză de tip cu componentele expresiei până ajunge la componente elementare  
**Siguranță de tip:** progres (expresia bine tipată nu se blochează), conservare (rămâne bine tipată după evaluare)  
**Unificare:** cea mai generală substituție conform căreia 2 expresii de tip devin identice  
**O variabilă unifică cu:** orice, cât timp occurs check este OK  
**2 constante unifică:** dacă sunt identice  
**2 construcții unifică:** dacă folosesc același constructor de tip pe argumente care unifică la rândul lor  
**Structuri folosite de algoritmul de unificare:** stiva de constrângeri, substituția

87

87