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 2.5 + 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")
  - **Constructori 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** ⇔ există un tip T astfel încât e :: T

Regulă de sinteză de tip

$$\frac{P_1,\ P_2,\ \ldots\ P_n}{C}\ (\text{id\_regulă})$$

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

C = concluzie (o relație de tipare)

7

7

Reguli de sinteză de tip – Exemple

$$\frac{}{\text{True, False} :: \text{Bool}}\ (TBool)$$

$$\frac{expr1 :: \text{Bool} \quad expr2 :: \text{a} \quad expr3 :: \text{a}}{\text{if } expr1 \text{ then } expr2 \text{ else } expr3 :: \text{a}}\ (TIf)$$

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

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

$$\frac{}{0, 1, 2\ \ldots :: \text{Int}}\ (TInt)$$

$$\frac{expr1 :: \text{Int} \quad expr2 :: \text{Int}}{expr1 + expr2 :: \text{Int}}\ (T+)$$

funcții

valori booleene

întregi

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` ⇔ `f = \g -> g 3 + 1`

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

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

b = Int

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

a = c ->Int

$$\frac{}{0, 1, 2\ \ldots :: \text{Int}}\ (TInt) \qquad \frac{}{1, 3 :: \text{Int}}\ (TInt)$$

c = Int

Așadar `f :: a -> b` ⇔ `f :: (c -> Int) -> Int` ⇔ `f :: (Int -> Int) -> Int`

Constrângeri

11

11

Sinteză de tip – Exemple

Care este tipul funcției fix: `fix f = f (fix f)` ⇔ `fix = \f -> f (fix f)`

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

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

a = c ->b

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

a = d, b = c

Așadar `fix :: a -> b` ⇔ `fix :: (c -> b) -> b` ⇔ `fix :: (b -> b) -> b`

Constrângeri

12

12

Antrenament: Expresii și signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

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

13

13

Antrenament: Expresii și signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

(++)	[a] -> [b] -> [c]
zipWith (ex: zipWith (+) [1,2] [3,4] = [4,6])	(a -> a -> a) -> [a] -> [a] -> [a]

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

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

14

14

Antrenament: Expresii și signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

zipWith (ex: zipWith (+) [1,2] [3,4] = [4,6])	(a -> a -> a) -> [a] -> [a] -> [a]
unzip (ex: unzip [(1,2), (3,4)] = ([1,3], [2,4]))	[(a,b)] -> ([a], [b])

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 signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

unzip (ex: unzip [(1,2), (3,4)] = ([1,3], [2,4]))	[(a,b)] -> ([a], [b])
[take, drop]	[Int -> [a] -> [a]]

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

16

16

Antrenament: Expresii și signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

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

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

17

17

Antrenament: Expresii și signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura 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 signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

Da:

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

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

(odd . fst)	(Int, b) -> Bool
map ("01" :) . reverse	[[Char]] -> [[Char]]

19

19

Antrenament: Expresii și signaturi

Exercițiu: Pentru expresia din stânga este corectă signatura din dreapta?

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

"01" : element înseamnă că "01" are tipul a și element are tipul [a].

Ex: (map ("01":) . reverse) [["So", "many"], ["Strings"]]

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

map ("01" :) . 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

Sinteză de tip – Cuprins

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

23

23

Unificare

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

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

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 -> [d] și a = Int -> c
  → b -> [d] trebuie să unifice cu Int -> c, altfel sinteza de tip va eșua
- Există S = {Int / b, [d] / c} a.î. b -> [d] și Int -> c se intersectează în Int -> [d]
- Există și S' = {Int / b, [Int] / c, Int / d}, rezultând în Int -> [Int], dar S este **mai generală**!

24

24

Mai multe exemple

- e1 = a, e2 = Int -> Bool
- e1 = (a, [a]), e2 = ([b], c)
- e1 = Int, e2 = Float

25

25

Mai multe exemple

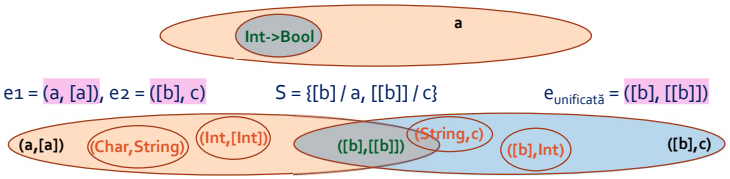
- e1 = a, e2 = Int -> Bool S = {Int -> Bool / a} $e_{unificată}$ = Int -> Bool

- e1 = (a, [a]), e2 = ([b], c)
- e1 = Int, e2 = Float

26

Mai multe exemple

- e1 = a, e2 = Int -> Bool S = {Int -> Bool / a} $e_{unificată}$ = Int -> Bool



- e1 = (a, [a]), e2 = ([b], c) S = {[b] / a, [[b]] / c} $e_{unificată}$ = ([b], [[b]])

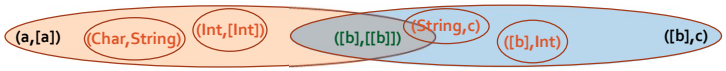
- e1 = Int, e2 = Float

27

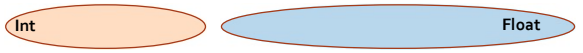
Mai multe exemple

- e1 = a, e2 = Int -> Bool S = {Int -> Bool / a} $e_{unificată}$ = Int -> Bool

- e1 = (a, [a]), e2 = ([b], c) S = {[b] / a, [[b]] / c} $e_{unificată}$ = ([b], [[b]])



- e1 = Int, e2 = Float Nu unifică (nu se fac conversii implicite)!



28

28

Unificare la sinteza de tip – Exemplu

Care este tipul funcției f: `f x = x x` ⇔ `f = \x -> x x`

Constrângeri

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

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

`a = c -> b = c`

a unifică cu c -> b, dar c -> b nu unifică cu c.
(c = c -> b = c -> b -> b = c -> b -> b -> b ... 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: t1 ~ t1 -> t

29

29

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 ($S = S_1 \cup \ldots \cup S_n$) dacă și numai dacă sunt aplicații ale aceluiași constructor de tip asupra câte n argumente care unifică recursiv (prin substituțiile $S_1 \ldots S_n$)

30

30

Unificare la sinteza de tip – Implementare

- Algoritmul de sinteză de tip pe bază de unificare folosește
  - O **stivă de constrângeri** de forma `e1 = e2` – 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 `e1 = e2` din stivă
  - Dacă `e1` unifică cu `e2` prin substituția S_{12}, se aplică S_{12}
    - asupra stivei și
    - asupra substituției S
  - Altfel, unificarea eșuează → sinteza de tip eșuează → eroare de tip

31

31

Exemplu

Stiva	Substituția

`a = b -> c`
`a = d -> d`
`b = Int`	

`b -> c = d -> d` `b = Int`	`a = b -> c`

32

32

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

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

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

36

36

Sinteză de tip – Cuprins

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

37

37

Determinarea tipului expresiilor / funcțiilor

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

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

(TLen) ⟶ []++[1,2] :: [a], []++"abcd" :: [b]

(T++) ⟶ [] :: [a], [1,2] :: [a], [] :: [b], "abcd" :: [b]

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

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

[] :: [Int] și [] :: [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+.

38

38

Determinarea tipului expresiilor / funcțiilor

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

39

39

Determinarea tipului expresiilor / funcțiilor

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

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

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

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]	
(T++) ⟶ xs :: [a], [1,2] :: [a], xs :: [b], "abcd" :: [b]	[a] = [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]
(T[]) ⟶ 1,2 :: a, 'a','b','c','d' :: 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	
(TInt, TChar) ⟶ 1,2 :: Int, 'a','b','c','d' :: Char	a = Int, b = Char

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

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.

45

45

Determinarea tipului expresiilor / funcțiilor

curry id	Constrângeri

46

46

Determinarea tipului expresiilor / funcțiilor

curry id	Constrângeri
(TApp) ⟶ expr :: b ⇔ curry :: a -> b, id :: a	

47

47

Determinarea tipului expresiilor / funcțiilor

curry id	Constrângeri
(TApp) ⟶ expr :: b ⇔ curry :: a -> b, id :: a	
$\overline{\text{curry} :: ((c, d) \to e) \to c \to d \to e}\ (TCurry)$	a = (c,d) -> e, b = c -> d -> e

48

48

Determinarea tipului expresiilor / funcțiilor

curry id

Constrângeri

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

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

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

a = (c,d) -> e, b = c -> d -> e

a = f -> f

Determinarea tipului expresiilor / funcțiilor

curry id

Constrângeri

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

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

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

a = (c,d) -> e, b = c -> d -> e

a = f -> f

În etapa de unificare: a = (c,d) -> e = f -> f ⟶ e = f = (c,d) ⟶ b = c -> d -> (c,d), deci expresia are tipul c -> d -> (c,d) conform TApp.

Determinarea tipului expresiilor / funcțiilor

uncurry id

Constrângeri

Determinarea tipului expresiilor / funcțiilor

uncurry id

Constrângeri

(TApp) ⟶ expr :: b ⇔ uncurry :: a -> b, id :: a

Determinarea tipului expresiilor / funcțiilor

uncurry id

(TApp) ⟶ expr :: b ⇔ uncurry :: a -> b, id :: a

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

Constrângeri

a = c -> d -> e, b = (c,d) -> e

53

Determinarea tipului expresiilor / funcțiilor

uncurry id

(TApp) ⟶ expr :: b ⇔ uncurry :: a -> b, id :: a

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

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

Constrângeri

a = c -> d -> e, b = (c,d) -> e

a = f -> f

54

Determinarea tipului expresiilor / funcțiilor

uncurry id

(TApp) ⟶ expr :: b ⇔ uncurry :: a -> b, id :: a

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

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

Constrângeri

a = c -> d -> e, b = (c,d) -> e

a = f -> f

În etapa de unificare: a = c -> d -> e = f -> f ⟶ c = f = d -> e ⟶ b = (d -> e, d) -> e, deci expresia are tipul (d -> e, d) -> e conform TApp.

55

Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

56

Determinarea tipului expresiilor / funcțiilor

curry (curry id) | Constrângeri

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

57

57

Determinarea tipului expresiilor / funcțiilor

curry (curry id) | Constrângeri

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

(ExAnterior) ⟶ curry id :: f -> g -> (f,g) | a = f -> g -> (f,g)

58

58

Determinarea tipului expresiilor / funcțiilor

curry (curry id) | Constrângeri

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

(ExAnterior) ⟶ curry id :: f -> g -> (f,g) | a = f -> g -> (f,g)

$$\frac{}{\text{curry} :: ((c, d) \to e) \to c \to d \to e} \ (TCurry)$$ | a = (c,d) -> e, b = c -> d -> e

59

59

Determinarea tipului expresiilor / funcțiilor

curry (curry id) | Constrângeri

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

(ExAnterior) ⟶ curry id :: f -> g -> (f,g) | a = f -> g -> (f,g)

$$\frac{}{\text{curry} :: ((c, d) \to e) \to c \to d \to e} \ (TCurry)$$ | a = (c,d) -> e, b = c -> d -> e

În etapa de unificare: a = f -> g -> (f,g) = (c,d) -> e ⟶ f = (c,d), e = g -> (f,g) = g -> ((c,d),g) ⟶ b = c -> d -> g -> ((c,d), g), deci expresia are tipul c -> d -> g -> ((c,d), g) conform TApp.

60

60

Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

61

Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

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

62

Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

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

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

a = (c -> d -> e) -> (c,d) -> e

63

Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

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

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

$$\frac{}{\text{curry} :: ((f,g) \rightarrow h) \rightarrow f \rightarrow g \rightarrow h} \ (TCurry)$$

a = (c -> d -> e) -> (c,d) -> e

a = (f,g) -> h, b = f -> g -> h

64

Determinarea tipului expresiilor / funcțiilor

curry uncurry

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

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

$$\frac{}{\text{curry} \;::\; ((f,g) \to h) \to f \to g \to h} \;(TCurry)$$

Constrângeri

a = (c -> d -> e) -> (c,d) -> e

a = (f,g) -> h, b = f -> g -> h

În etapa de unificare: a = (c -> d -> e) -> (c,d) -> e = (f,g) -> h → (printre altele) c -> d -> 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).

65

Sinteză de tip – Cuprins

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

66

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 `->'

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

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

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
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 = dinainte de |

68

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                                   Not in scope: `cons'
      | null x = cons a x
      | otherwise = f (a+1) (tail x)

f a x                                   Not in scope: `oterwise'
      | null x = a                      Perhaps you meant `otherwise'
      | oterwise = f (a+1) (tail x)

f x = sort (take 10 x)                  Not in scope: `sort'
```

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                                   Not in scope: `cons'
      | null x = cons a x               Înlocuiește cu (:)
      | otherwise = f (a+1) (tail x)

f a x                                   Not in scope: `oterwise'
      | null x = a                      Perhaps you meant `otherwise'
      | oterwise = f (a+1) (tail x)

f x = sort (take 10 x)                  Not in scope: `sort'
```

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

70

70

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
```

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
                                Șterge (<)
f x = if null x then 0 else 1 + f tail x
                                ...
                                Probable cause: `tail' is applied to too few arguments
                                Pune paranteze la tail: f (tail x)
```

72

72

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

```
f = 'A' ++ "BC"
```
Couldn't match expected type `[Char]' with actual type `Char'

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

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
```
Couldn't match type `Char' with `[Char]'
Fie f :: String -> Char, fie f = (: []) . head

```
f = 'A' ++ "BC"
```
Couldn't match expected type `[Char]' with actual type `Char'
Fie 'A' : "BC", fie "A" ++ "BC"

```
f = (:) "BC" 'A'
```
Couldn't match expected type `[[Char]]' with actual type `Char'
In the second argument of `(:)', namely 'A'
Schimb ordinea: (:) 'A' "BC"

74

74

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
```
*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)

```
mySum [] = 0
mySUm (x:xs) = x + mySum xs
```
*Main> mySum [1..5]
*** Exception: ... Non-exhaustive patterns ... mySum

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

```
mySum [] = 0
mySUm (x:xs) = x + mySum xs
```
*Main> mySum [1..5]
*** Exception: ... Non-exhaustive patterns ... mySum
Corectează mySUm în mySum

76

76

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

77

77

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

78

78

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

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: 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

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: 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

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: 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

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: orice, cât timp occurs check este OK
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: dacă sunt identice
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: dacă folosesc același constructor de tip pe argumente care unifică la rândul lor
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: stiva de constrângeri, substituția

86

86