

PARADIGME DE PROGRAMARE

Curs 15

Sinteză de tip.

Sinteză de tip – Cuprins

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

Sistem de tipuri

Sistem de tipuri = set de mecanisme și reguli privind organizarea, construirea, manevrarea tipurilor de date acceptate în limbaj

Atribuții posibile

- **Asociere:** asignează un tip expresiilor
- **Definire:** permite crearea de noi tipuri (TDA-uri)
- **Compatibilitate:** stabilește reguli de echivalență/compatibilitate de tip
- **Sinteză (inferență):** determină automat tipurile, când ele nu sunt specificate explicit

Avantaje și limitări

Avantaje

- **Siguranță:** detectează erorile și codul invalid încă din faza de compilare
- **Claritate:** declarațiile de tip documentează forma cea mai generală a funcției (și asigură consistența cu modificările ulterioare)
- **Eficiență:** permite compilatorului optimizări specifice pe tipuri

Limitări

- **Rigiditate:** Respinge cod neproblematic, care nu respectă regulile sintactice
Restricționează funcțiile / structurile de date (ex: lipsa listelor eterogene)
- **Efort:** Necesită declarații de tip sau mecanisme complexe de sinteză de tip

Sinteză de tip – Cuprins

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

Sinteză (inferență) de tip

Sinteză de tip = **determinarea automată a tipului unei expresii, pe baza regulilor limbajului (reguli de sinteză de tip)**

- Tipul unei expresii este sintetizat în funcție de:
 - **Tipul componentelor:** ex: $2.5+2 :: \text{Fractional } a \Rightarrow a$
 - **Context lexical:** ex: în contextul ['a' .. x], $x :: \text{Char}$
- Tipul sintetizat este o expresie de tip care combină:
 - **Constante de tip:** tipuri primitive: Bool, Char, Int, etc.
 - **Variabile de tip:** tipuri generice (orice tip): a, b, etc.
 - **Constructorii de tip:** (,) [] -> constructori definiți cu „data”

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: descrie cum deducem un tip nou din premise existente

$$\frac{P_1, P_2, \dots, P_n}{C} \text{ (id_regulă)}$$

P_i = premise (relații de tipare deja cunoscute)

id_regulă = identificator unic al regulii aplicate

C = concluzie (relația de tipare dedusă)

Reguli de sinteză de tip – Exemple

$$\frac{}{\text{True, False} :: \text{Bool}} \quad (T\text{Bool})$$
$$\frac{\text{expr1} :: \text{Bool} \quad \text{expr2} :: a \quad \text{expr3} :: a}{\text{if expr1 then expr2 else expr3} :: a} \quad (T\text{If})$$
$$\frac{}{0, 1, 2 \dots :: \text{Int}} \quad (T\text{Int})$$
$$\frac{\text{expr1} :: \text{Int} \quad \text{expr2} :: \text{Int}}{\text{expr1} + \text{expr2} :: \text{Int}} \quad (T+)$$
$$\frac{\text{var} :: a \quad \text{expr} :: b}{\backslash \text{var} \rightarrow \text{expr} :: a \rightarrow b} \quad (T\text{Abs})$$
$$\frac{\text{fun} :: a \rightarrow b \quad \text{expr} :: a}{\text{fun expr} :: b} \quad (T\text{App})$$

funcții

valori booleene

întregi

Observații

- Tipul unei expresii este unic
- Regulile de tipare funcționează în ambele sensuri (în sinteza de tip pornim de la concluzie către premise)
 - De la premise către concluzie
Dacă $\text{expr}_1 :: \text{Int}$ și $\text{expr}_2 :: \text{Int}$, atunci $\text{expr}_1 + \text{expr}_2 :: \text{Int}$
 - De la concluzie către premise
Dacă $\text{expr}_1 + \text{expr}_2 :: \text{Int}$, atunci $\text{expr}_1 :: \text{Int}$ și $\text{expr}_2 :: \text{Int}$

Algoritm de sinteză de tip

- **Identifică regula aplicabilă**, prin pattern matching între
 - Expresia care trebuie tipată
 - Concluziile regulilor de sinteză de tip
- **Aplică regula**, obținând constrângerile asupra componentelor expresiei
- **Recursivitate**: reia algoritmul pentru fiecare componentă, până când nu mai rămâne nimic de descompus în părți componente
- **Determină tipul expresiei** prin aplicarea tuturor constrângerilor acumulate în pașii anteriori

Sinteză de tip – Exemple

Care este tipul funcției f: $fg = g3 + 1 \Leftrightarrow f = \lambda g \rightarrow g3 + 1$

Constrângeri

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

$$\frac{g :: a \quad g3 + 1 :: b}{f :: a \rightarrow b} \quad (TAbs)$$

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

$$\frac{g3 :: Int \quad 1 :: Int}{g3 + 1 :: Int} \quad (T+)$$

$b = Int$

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

$$\frac{g :: c \rightarrow Int \quad 3 :: c}{g3 :: Int} \quad (TApp)$$

$a = c \rightarrow Int$

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

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

$c = Int$

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

Sinteză de tip – Exemple

Care este tipul funcției fix: $\text{fix } f = f (\text{fix } f) \iff \text{fix} = \lambda f \rightarrow f (\text{fix } f)$

Constrângeri

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

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

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

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

$$a = c \rightarrow b$$

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

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

$$a = d, b = c$$

Așadar $\text{fix} :: a \rightarrow b \iff \text{fix} :: (c \rightarrow b) \rightarrow b \iff \text{fix} :: (b \rightarrow b) \rightarrow b$

Antrenament: Expresii și semnături

Pentru expresia din stânga este corectă semnătura din dreapta?

`(++)`

`zipWith` (ex: `zipWith (+) [1,2] [3,4] = [4,6]`)

`unzip` (ex: `unzip [(1,2), (3,4)] = ([1,3], [2,4])`)

`[take, drop]`

`words` (ex: `words "wait for it" = ["wait", "for", "it"]`)

`(odd . fst)`

`map ("o1" :) . reverse`

`[a] -> [b] -> [c]`

`(a -> a -> a) -> [a] -> [a] -> [a]`

`[(a,b)] -> ([a], [b])`

`[Int -> [a] -> [a]]`

`String -> String`

`(Int, b) -> Bool`

`[[Char]] -> [[Char]]`

Antrenament: Expresii și semnături

Pentru expresia din stânga este corectă semnătura 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]`

Antrenament: Expresii și semnături

Pentru expresia din stânga este corectă semnătura 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]`

Antrenament: Expresii și semnături

Pentru expresia din stânga este corectă semnătura 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')]`

Antrenament: Expresii și semnături

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

[take, drop]

[Int -> [a] -> [a]]

words (ex: words "wait for it" = ["wait", "for", "it"])

String -> String

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

Antrenament: Expresii și semnături

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`

Antrenament: Expresii și semnături

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

Antrenament: Expresii și semnături

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]]`

Sinteză de tip – Cuprins

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

Siguranță de tip

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

Proprietăți fundamentale

(garantează că expresiile bine tipate „nu o pot lua pe căi greșite” în timpul evaluării):

- **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 $e' :: T$

Sinteză de tip – Cuprins

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

Unificare

Unificarea a 2 expresii de tip = **găsirea celei mai generale substituții $S = \{e_1 / t_1, \dots, e_n / t_n\}$ pentru variabilele de tip t_i din cele 2 expresii astfel încât, în urma substituției, cele 2 expresii de tip devin identice**

- Se folosește în sinteza de tip, în pasul de **aplicare a constrângerilor**, pentru a determina dacă două expresii pot avea același tip

Exemplu: Constrângeri rezultate din sinteza de tip: $a = b \rightarrow [d]$ $a = \text{Int} \rightarrow c$

- $b \rightarrow [d]$ trebuie să unifice cu $\text{Int} \rightarrow c$, altfel sinteza de tip eșuează
- Există $S = \{\text{Int} / b, [d] / c\}$ a.î. $b \rightarrow [d]$ și $\text{Int} \rightarrow c$ se intersectează în $\text{Int} \rightarrow [d]$
- Există și $S' = \{\text{Int} / b, [\text{Int}] / c, \text{Int} / d\}$ expresia unificată este $\text{Int} \rightarrow [\text{Int}]$, dar S este mai generală!

Mai multe exemple

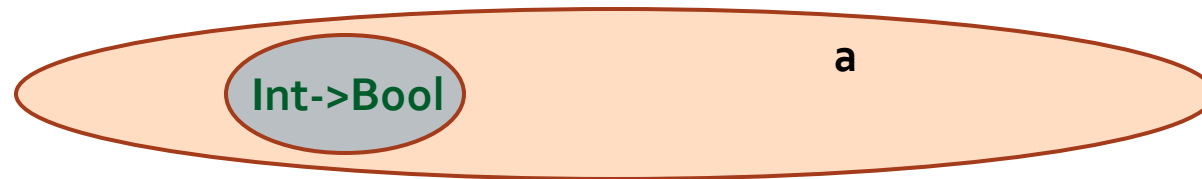
- $e_1 = a$, $e_2 = \text{Int} \rightarrow \text{Bool}$
- $e_1 = (a, [a])$, $e_2 = ([b], c)$
- $e_1 = \text{Int}$, $e_2 = \text{Float}$

Mai multe exemple

- $e_1 = a$, $e_2 = \text{Int} \rightarrow \text{Bool}$

$S = \{\text{Int} \rightarrow \text{Bool} / a\}$

$e_{\text{unificată}} = \text{Int} \rightarrow \text{Bool}$



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

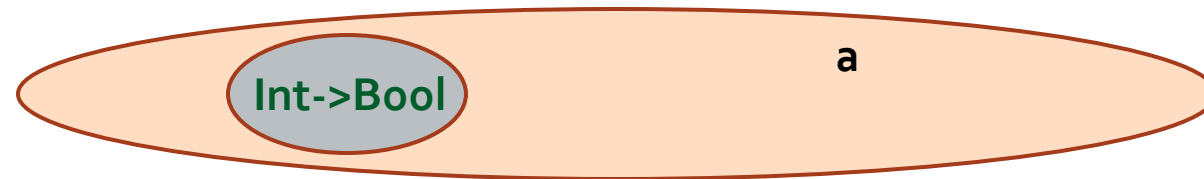
- $e_1 = \text{Int}$, $e_2 = \text{Float}$

Mai multe exemple

- $e_1 = a$, $e_2 = \text{Int} \rightarrow \text{Bool}$

$S = \{\text{Int} \rightarrow \text{Bool} / a\}$

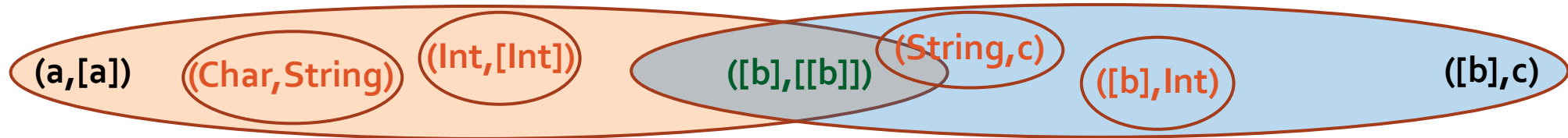
$e_{\text{unificată}} = \text{Int} \rightarrow \text{Bool}$



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

$S = \{[b] / a, [[b]] / c\}$

$e_{\text{unificată}} = ([b], [[b]])$



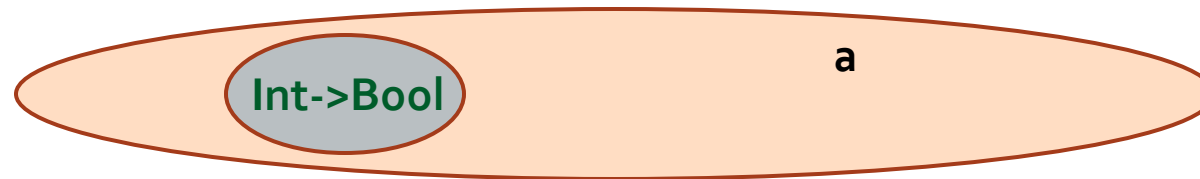
- $e_1 = \text{Int}$, $e_2 = \text{Float}$

Mai multe exemple

• $e_1 = a$, $e_2 = \text{Int} \rightarrow \text{Bool}$

$S = \{\text{Int} \rightarrow \text{Bool} / a\}$

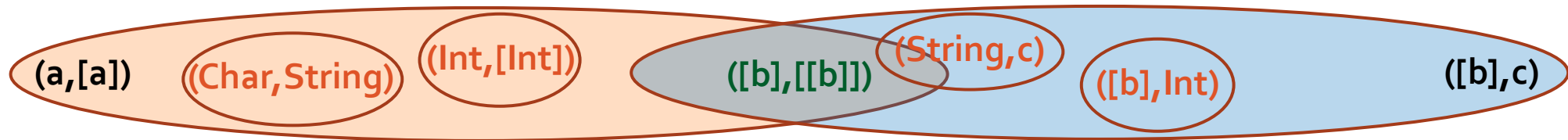
$e_{\text{unificată}} = \text{Int} \rightarrow \text{Bool}$



• $e_1 = (a, [a])$, $e_2 = ([b], c)$

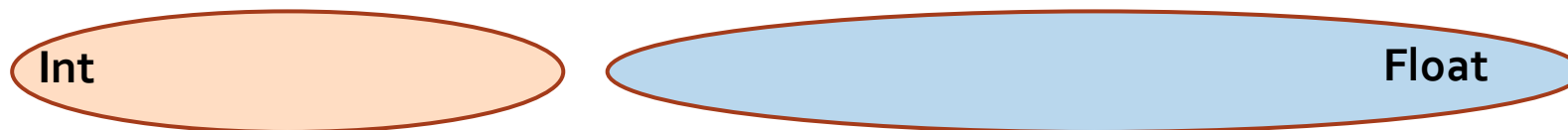
$S = \{[b] / a, [[b]] / c\}$

$e_{\text{unificată}} = ([b], [[b]])$



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

Nu unifică (nu se fac conversii implicite)!



Exemplu de eșec la unificare

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

Constrângeri

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

$$\frac{x :: a \quad xx :: b}{f :: a \rightarrow b} (TAbs)$$

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

$$\frac{x :: c \rightarrow b \quad x :: c}{xx :: b} (TApp)$$

$$a = c \rightarrow b = c$$

- a unifică cu $c \rightarrow b$, dar $c \rightarrow b$ nu unifică cu c
 - $c = c \rightarrow b = (c \rightarrow b) \rightarrow b = ((c \rightarrow b) \rightarrow b) \rightarrow b \dots$ tip infinit, care nu poate fi sintetizat

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

Reguli 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
 - argumentele constructorului de tip unifică fiecare cu fiecare (prin substituțiile $S_1 \dots S_n$)

Unificare la sinteza de tip – Implementare

- Algoritmul de sinteză de tip pe bază de unificare folosește două structuri de date:
 - **stiva de constrângeri**: constrângeri $e1 = e2$ acumulate la aplicarea regulilor de sinteză de tip
 - **substituția (S)**: mulțime (inițial vidă) de asocieri **variabilă de tip = expresie de tip**
- Fiecare iterație a algoritmului de aplicare a constrângerilor
 - **Extrage** o constrângere $e1 = e2$ din stivă
 - **Unifică** $e1$ cu $e2$
 - Succes → substituția rezultată din unificare se aplică asupra celor 2 structuri de date
 - Eșec → eroare de tip

Exemplu

| Stiva | Substituția |
|--|----------------------------|
| <code>a = b -> c</code> <code>a = d -> d</code> <code>b = Int</code> | |
| <code>b -> c = d -> d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| | |
| | |
| | |
| | |

Exemplu

| Stiva | Substituția |
|--|----------------------------|
| <code>a = b -> c</code> <code>a = d -> d</code> <code>b = Int</code> | |
| <code>b -> c = d -> d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>b = d</code> <code>c = d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| | |
| | |
| | |

Exemplu

| Stiva | Substituția |
|--|--|
| <code>a = b -> c</code> <code>a = d -> d</code> <code>b = Int</code> | |
| <code>b -> c = d -> d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>b = d</code> <code>c = d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>c = d</code> <code>d = Int</code> | <code>a = d -> c</code> <code>b = d</code> |
| | |
| | |

Exemplu

| Stiva | Substituția |
|--|--|
| <code>a = b -> c</code> <code>a = d -> d</code> <code>b = Int</code> | |
| <code>b -> c = d -> d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>b = d</code> <code>c = d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>c = d</code> <code>d = Int</code> | <code>a = d -> c</code> <code>b = d</code> |
| <code>d = Int</code> | <code>a = d -> d</code> <code>b = d</code> <code>c = d</code> |
| | |

Exemplu

| Stiva | Substituția |
|--|--|
| <code>a = b -> c</code> <code>a = d -> d</code> <code>b = Int</code> | |
| <code>b -> c = d -> d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>b = d</code> <code>c = d</code> <code>b = Int</code> | <code>a = b -> c</code> |
| <code>c = d</code> <code>d = Int</code> | <code>a = d -> c</code> <code>b = d</code> |
| <code>d = Int</code> | <code>a = d -> d</code> <code>b = d</code> <code>c = d</code> |
| | <code>a = Int -> Int</code> <code>b = Int</code> <code>c = Int</code> <code>d = Int</code> |

Sinteză de tip – Cuprins

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

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)

Unificare: `expr :: Int`

- nu există constrângeri multiple asupra aceleiași variabile

Determinarea tipului expresiilor / funcțiilor

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

Constrângeri

Determinarea tipului expresiilor / funcțiilor

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

Constrângeri

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

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]`

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]`

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`

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`

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`

Unificare: eroare de tip

- `[Int] = [Char]` → `Int = Char` nu unifică

Determinarea tipului expresiilor / funcțiilor

curry id

Constrângeri

Determinarea tipului expresiilor / funcțiilor

curry id

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

curry id

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

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

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

curry id

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

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

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

Constrângeri

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

$a = f \rightarrow f$

Determinarea tipului expresiilor / funcțiilor

curry id

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

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

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

Unificare: curry id $::$ c \rightarrow d \rightarrow (c,d)

- a = (c,d) \rightarrow e = f \rightarrow f \rightarrow
e = f = (c,d) \rightarrow
b = c \rightarrow d \rightarrow (c,d)

Constrângeri

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

a = f \rightarrow f

Determinarea tipului expresiilor / funcțiilor

uncurry id

Constrângeri

Determinarea tipului expresiilor / funcțiilor

uncurry id

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

uncurry id

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

$$\frac{}{\text{uncurry} :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e} \quad (T\text{Uncurry})$$

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

uncurry id

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

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

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

Constrângeri

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

$a = f \rightarrow f$

Determinarea tipului expresiilor / funcțiilor

uncurry id

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

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

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

Unificare: $\text{uncurry id} :: (d \rightarrow e, d) \rightarrow e$

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

Constrângeri

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

$a = f \rightarrow f$

Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

Determinarea tipului expresiilor / funcțiilor

curry (curry id)

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

curry (curry id)

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

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

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

curry (curry id)

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

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

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

Constrângeri

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

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

Determinarea tipului expresiilor / funcțiilor

curry (curry id)

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

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

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

Unificare: $\text{curry (curry id)} :: c \rightarrow d \rightarrow g \rightarrow ((c,d), g)$

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

Constrângeri

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

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

Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

Determinarea tipului expresiilor / funcțiilor

curry uncurry

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

curry uncurry

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

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

Constrângeri

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

Determinarea tipului expresiilor / funcțiilor

curry uncurry

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

$$\frac{}{\text{uncurry} :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e} \quad (T\text{Uncurry})$$

$$\frac{}{\text{curry} :: ((f, g) \rightarrow h) \rightarrow f \rightarrow g \rightarrow h} \quad (T\text{Curry})$$

Constrângeri

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

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

Determinarea tipului expresiilor / funcțiilor

curry uncurry

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

$$\frac{}{\text{uncurry} :: (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e} \quad (T\text{Uncurry})$$

$$\frac{}{\text{curry} :: ((f, g) \rightarrow h) \rightarrow f \rightarrow g \rightarrow h} \quad (T\text{Curry})$$

Constrângeri

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

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

Unificare: eroare de tip

- $a = (c \rightarrow d \rightarrow e) \rightarrow (c, d) \rightarrow e = (f, g) \rightarrow h \rightarrow$
 $c \rightarrow d \rightarrow e = (f, g)$ nu unifică (sunt aplicații ale unor constructori de tip diferiți)

Alte exerciții: curry curry, uncurry curry, uncurry uncurry (nu toate dau eroare).

Sinteză de tip – Cuprins

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

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 `|'

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
```

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

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

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

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

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

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`

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) și vezi ce modul trebuie importat (aici `import Data.List`)

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

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)

Couldn't match expected type with actual type

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

```
f :: String -> String    Couldn't match type `Char' with `[Char]'
```

```
f = head
```

```
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'

Couldn't match expected type with actual type

- fie nu suntem în concordanță cu semnatura (ori corectăm funcția, ori semnatura)
- 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"`

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

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

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)
```

Adaugă | otherwise = ...

```
mySum [] = 0
```

```
mySUM (x:xs) = x + mySum xs
```

```
*Main> mySum [1..5]
```

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

Corectează mySUM în mySum

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

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

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

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

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

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

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

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

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

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