Curs 14

2021-2022

Fundamentele limbajelor de programare

Cuprins

1. Determinarea tipurilor
   - Asociere de tipuri

Sintaxa limbajului LAMBDA

BNF

$$
\begin{array}{rcl}
e & ::= & x \mid n \mid \mathit{true} \mid \mathit{false} \\
 & \mid & e + e \mid e < e \mid \texttt{not}\,(e) \\
 & \mid & \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
 & \mid & \lambda x.e \mid e\ e \\
 & \mid & \texttt{let}\ x = e\ \texttt{in}\ e
\end{array}
$$

Verificarea sintaxei în Prolog

```
exp(Id) :- atom(Id).                             % identificator
exp(Lit) :- Lit = true ; Lit = false ; integer(Lit).
exp(E1 + E2) :- exp(E1), exp(E2).
exp(if(E1, E2, E3)) :- exp(E1), exp(E2), exp(E3).
exp(Id -> Exp) :- atom(Id), exp(Exp).            % lambda
exp(Exp1 $ Exp2) :- exp(Exp1), exp(Exp2).        % aplicare
exp(let(Id, Exp1, Exp2)) :- atom(Id), exp(Exp1), exp(Exp2).
```

Semantica small-step pentru Lambda

Prolog

```
step(Env, X, V) :- atom(X), get(Env, X, V).
step(_, I1 + I2,I):- integer(I1),integer(I2),
                     I is I1 + I2.
step(Env, AE + AE1,AE + AE2):- step(Env, AE1,AE2).
step(Env, AE1 + AE,AE2 + AE):- step(Env, AE1,AE2).
step(Env, X -> E, closure(X, E, Env)).
step(_, let(X, E1, E2), (X -> E2) $ E1).
step(Env, E $ E1, E $ E2) :- step(Env, E1, E2).
step(Env, E1 $ E, E2 $ E) :- step(Env, E1, E2).
step(Env, closure(X, E, EnvE) $ V, Result) :-
        set(EnvE, X, V, EnvEX),
        step(EnvEX, E, E1)
        ->  Result = closure(X, E1, EnvE) $ V
        ;   Result = E.
```

Problemă: Sintaxa este prea permisivă

Problemă: Mulți termeni acceptați de sintaxă nu pot fi evaluați

- 2 $(\lambda x.x)$— expresia din stânga aplicației trebuie să reprezinte o funcție
- $(\lambda x.x) + 1$ — adunăm funcții cu numere
- $(\lambda x.x + 1)$ $(\lambda x.x)$ — pot face o reducție, dar tot nu pot evalua

```
?- run((x -> x) +1, V).
V = closure(x, x, [])+1.

?- run(2 $ (x -> x), V).
V = 2$closure(x, x, []).
```

Problemă: Sintaxa este prea permisivă

Problemă: Mulți termeni acceptați de sintaxă nu pot fi evaluați

- $2\ (\lambda x.x)$— expresia din stânga aplicației trebuie să reprezinte o funcție
- $(\lambda x.x) + 1$ — adunăm funcții cu numere
- $(\lambda x.x + 1)\ (\lambda x.x)$ — pot face o reducție, dar tot nu pot evalua

Soluție: Identificarea (precisă) a programelor corecte

- Definim tipuri pentru fragmente de program corecte (e.g., int, bool)
- Definim (recursiv) o relație care să lege fragmente de program de tipurile asociate

$$((\lambda x.x + 1)\ ((\lambda x.x)\ 3)) : \text{int}$$

Relația de asociere de tipuri

Definim (recursiv) o relație de forma $\Gamma \vdash e : \tau$, unde

- τ este un tip

$$
\begin{aligned}
\tau ::= \; & int \text{ [întregi]} \\
\mid \; & bool \text{ [valori de adevăr]} \\
\mid \; & \tau \to \tau \text{ [funcții]} \\
\mid \; & a \text{ [variabile de tip]}
\end{aligned}
$$

- e este un termen (potențial cu variabile libere)
- Γ este mediul de tipuri, o funcție parțială finită care asociază tipuri variabilelor (libere ale lui e)
- Variabilele de tip sunt folosite pentru a indica polimorfismul

Cum citim $\Gamma \vdash e : \tau$?

Dacă variabila x are tipul $\Gamma(x)$ pentru orice $x \in dom(\Gamma)$,
atunci termenul e are tipul τ.

Axiome

$(:\text{VAR}) \quad \Gamma \vdash x : \tau \quad \textit{dacă}\ \Gamma(x) = \tau$

$(:\text{INT}) \quad \Gamma \vdash n : \textit{int} \quad \textit{dacă}\ n\ \text{întreg}$

$(:\text{BOOL}) \quad \Gamma \vdash b : \textit{bool} \quad \textit{dacă}\ b = \textit{true}\ \textit{or}\ b = \textit{false}$

Expresii

$$(\text{:IOP})\quad \frac{\Gamma \vdash e_1 : \mathit{int} \quad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1 \ o\ e_2 : \mathit{int}} \quad \textit{dacă } o \in \{+, -, *, /\}$$

$$(\text{:COP})\quad \frac{\Gamma \vdash e_1 : \mathit{int} \quad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1 \ o\ e_2 : \mathit{bool}} \quad \textit{dacă } o \in \{\leq, \geq, <, >, =\}$$

$$(\text{:BOP})\quad \frac{\Gamma \vdash e_1 : \mathit{bool} \quad \Gamma \vdash e_2 : \mathit{bool}}{\Gamma \vdash e_1 \ o\ e_2 : \mathit{bool}} \quad \textit{dacă } o \in \{,\}$$

$$(\text{:IF})\quad \frac{\Gamma \vdash e_b : \mathit{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathtt{if}\ e_b\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \tau}$$

Fragmentul funcțional

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \quad \textit{dacă } \Gamma' = \Gamma[x \mapsto \tau]$$

$$(\text{:APP})\quad \frac{\Gamma \vdash e_1 : \tau' \to \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau}$$

Probleme computaționale

Verificarea tipului

Date fiind Γ, e și τ, verificați dacă $\Gamma \vdash e : \tau$.

Determinarea (inferarea) tipului

Date fiind Γ și e, găsiți (sau arătați ce nu există) un τ astfel încât $\Gamma \vdash e : \tau$.

- A doua problemă e mai grea în general decât prima
- Algoritmi de inferare a tipurilor
  - Colectează constrângeri asupra tipului
  - Folosesc metode de rezolvare a constrângerilor (programare logică)

Exemplu

Care este tipul expresiei următoare (dacă are)

$$\lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y$$

Aplicăm regula

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \quad \textit{dacă}\ \Gamma' = \Gamma[x \mapsto \tau]$$

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_x \to t$ dacă

$x \mapsto t_x \vdash \lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t$

Exemplu

Care este tipul expresiei următoare (dacă are)

$$\lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y$$

Aplicăm regula

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \quad \textit{dacă}\ \Gamma' = \Gamma[x \mapsto \tau]$$

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_x \to t$ dacă
$x \mapsto t_x \vdash \lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t$
Mai departe: $x \mapsto t_x \vdash \lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_y \to t_0$ dacă
$x \mapsto t_x, y \mapsto t_y \vdash \lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_0$ și, de mai sus,
$t = t_y \to t_0$

Exemplu

Care este tipul expresiei următoare (dacă are)

$$\lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y$$

Aplicăm regula

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \quad \textit{dacă}\ \Gamma' = \Gamma[x \mapsto \tau]$$

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_x \to t$ dacă
$x \mapsto t_x \vdash \lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t$
Mai departe: $x \mapsto t_x \vdash \lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_y \to t_0$ dacă
$x \mapsto t_x, y \mapsto t_y \vdash \lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_0$ și, de mai sus,
$t = t_y \to t_0$
Mai departe: $x \mapsto t_x, y \mapsto t_y \vdash \lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_z \to t_1$
dacă $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_1$ și, de mai
sus, $t_0 = t_z \to t_1$

Exemplu

Unde suntem

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_x \rightarrow t$ dacă
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_1$ și $t_0 = t_z \rightarrow t_1$,
$t = t_y \rightarrow t_0$.

Aplicăm regula (:IF) $\dfrac{\Gamma \vdash e_b : bool \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if}\ e_b\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}$

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_1$ dacă
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y = 0 :$bool și $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash z : t_1$ și
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x/y : t_1$

Exemplu

Aplicăm regula

$$(\text{:COP})\quad \frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 \ o \ e_2 : bool} \quad dacă\ o \in \{\leq, \geq, <, >, =\}$$

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y = 0$:bool dacă
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y$:int și $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash 0$:int

Aplicăm regula $(\text{:INT})\quad \Gamma \vdash n : int \quad dacă\ n$ *întreg*

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash 0$:int este adevărat

Aplicăm regula

$$(\text{:IOP})\quad \frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 \ o \ e_2 : int} \quad dacă\ o \in \{+, -, *, /\}$$

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x/y$:int dacă
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x$:int și $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y$:int
și, de mai sus, $t_1 =$int

Exemplu

Recapitulăm

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_x \to t$ dacă
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y : \texttt{int}$ și $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash z : t_1$
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x : \texttt{int}$ și $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y : \texttt{int}$
și $t_0 = t_z \to t_1$, $t = t_y \to t_0$, $t_1 = \texttt{int}$.

Aplicăm regula (:VAR) $\Gamma \vdash x : \tau$ *dacă* $\Gamma(x) = \tau$

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y : t_y$ adevărat și, de mai sus $t_y = \texttt{int}$
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash z : t_z$ adevărat și, de mai sus, $t_1 = t_z$
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x : t_x$ adevărat și, de mai sus, $t_x = \texttt{int}$

Exemplu

Finalizăm

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : t_x \to t$ dacă
$t_0 = t_z \to t_1$, $t = t_y \to t_0$, $t_1 = \texttt{int}$, $t_y = \texttt{int}$, $t_1 = t_z$ și $t_x = \texttt{int}$.

Rezolvăm constrângerile și obținem

$\vdash \lambda x.\lambda y.\lambda z.\ \texttt{if}\ y = 0\ \texttt{then}\ z\ \texttt{else}\ x/y : \texttt{int} \to \texttt{int} \to \texttt{int} \to \texttt{int}$

Relația de asociere de tipuri în Prolog

Definim (recursiv) o relație de forma `type(Gamma, E, T)`, unde

- `Gamma` este o listă de perechi de forma `(X, T)` unde `X` este un identificator și `T` este o expresie de tip cu variabile
- `E` este o λ-expresie scrisă cu sintaxa descrisă mai sus
- `T` este o expresie de tip cu variabile

Sintaxa limbajului LAMBDA si sintaxa tipurilor

BNF LAMBDA

$$
\begin{array}{rcl}
e & ::= & x \mid n \mid \mathit{true} \mid \mathit{false} \\
& \mid & e + e \mid e < e \mid \texttt{not}\,(e) \\
& \mid & \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
& \mid & \lambda x.e \mid e\ e \\
& \mid & \texttt{let}\ x = e\ \texttt{in}\ e
\end{array}
$$

BNF TIPURI

τ ::= *int* [întregi]
| *bool* [valori de adevăr]
| $\tau \to \tau$ [funcții]
| a [variabile de tip]

Sintaxa tipurilor

BNF

τ ::= *int* [întregi]
 | *bool* [valori de adevăr]
 | $\tau \to \tau$ [funcții]
 | *a* [variabile de tip]

Verificarea sintaxei tipurilor în Prolog

```
is_type(X) :- var(X).        % X este variabila neinstantiata
is_type(int).                % intregi
is_type(bool).               % valori de adevar
is_type(T1 -> T2) :-         % functii
   is_type(T1), is_type(T2).
```

Axiome

(:VAR) $\Gamma \vdash x : \tau$ *dacă* $\Gamma(x) = \tau$

```
type(Gamma, X, T) :- atom(X), get(Gamma, X, T).
```

(:INT) $\Gamma \vdash n : int$ *dacă* n întreg

```
type(_, I, int) :- integer(I).
```

(:BOOL) $\Gamma \vdash b : bool$ *dacă* $b = true$ *or* $b = false$

```
type(_, true, bool).
type(_, false, bool).
```

Expresii

$$(:\text{IOP})\quad \frac{\Gamma \vdash e_1 : \mathit{int} \quad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1\ o\ e_2 : \mathit{int}} \quad \textit{dacă } o \in \{+, -, *, /\}$$

```
type(Gamma, E1 + E2, int) :-
    type(Gamma, E1, int), type(Gamma, E2, int).
```

$$(:\text{COP})\quad \frac{\Gamma \vdash e_1 : \mathit{int} \quad \Gamma \vdash e_2 : \mathit{int}}{\Gamma \vdash e_1\ o\ e_2 : \mathit{bool}} \quad \textit{dacă } o \in \{\leq, \geq, <, >, =\}$$

```
type(Gamma, E1 < E2, bool) :-
    type(Gamma, E1, int), type(Gamma, E2, int).
```

$$(:\text{BOP})\quad \frac{\Gamma \vdash e_1 : \mathit{bool} \quad \Gamma \vdash e_2 : \mathit{bool}}{\Gamma \vdash e_1\ o\ e_2 : \mathit{bool}} \quad \textit{dacă } o \in \{,\}$$

```
type(Gamma, and(E1, E2), bool) :-
    type(Gamma, E1, bool), type(Gamma, E2, bool).
```

Expresia condițională

$$(\text{:IF})\quad \frac{\Gamma \vdash e_b : bool \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if}\ e_b\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}$$

```
type(Gamma, if(E, E1, E2), T) :-
    type(Gamma, E, bool),
    type(Gamma, E1, T),
    type(Gamma, E2, T).
```

Fragmentul funcțional

$$(:\text{FN}) \quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \lambda x.e : \tau \to \tau'} \quad \textit{dacă } \Gamma' = \Gamma[x \mapsto \tau]$$

```
type(Gamma, X -> E, TX -> TE) :-
                     atom(X),
                     set(Gamma, X, TX, GammaX),
                     type(GammaX, E, TE).
```

$$(:\text{APP}) \quad \frac{\Gamma \vdash e_1 : \tau' \to \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ e_2 : \tau}$$

```
type(Gamma, E1 $ E2, T) :-
                   type(Gamma, E, TE2 -> T),
                   type(Gamma, E2, TE2).
```

Tipurile variabile nu sunt suficiente

Tipurile variabile sunt destul de flexibile

- $\vdash \lambda x.x : t \to t$ pentru orice t
- $\vdash \mathtt{if}\ (\lambda x.x)\ \mathit{true}\ \mathtt{then}\ (\lambda x.x)\ 3\ \mathtt{else}\ 4 : \mathtt{int}$

Tipurile variabile nu sunt suficiente

Tipurile variabile sunt destul de flexibile

- $\vdash \lambda x.x : t \to t$ pentru orice t
- $\vdash \texttt{if } (\lambda x.x)\ \mathit{true}\ \texttt{then } (\lambda x.x)\ 3\ \texttt{else } 4 : \texttt{int}$

Dar tipul unei expresii este fixat:

$$\nvdash (\lambda id.\texttt{if } id\ \mathit{true}\ \texttt{then } id\ 3\ \texttt{else } 4)(\lambda x.x) : \texttt{int}$$

Tipurile variabile nu sunt suficiente

Tipurile variabile sunt destul de flexibile

- $\vdash \lambda x.x : t \to t$ pentru orice t
- $\vdash \texttt{if}\ (\lambda x.x)\ \mathit{true}\ \texttt{then}\ (\lambda x.x)\ 3\ \texttt{else}\ 4 : \texttt{int}$

Dar tipul unei expresii este fixat:

$$\nvdash (\lambda id.\texttt{if}\ id\ \mathit{true}\ \texttt{then}\ id\ 3\ \texttt{else}\ 4)(\lambda x.x) : \texttt{int}$$

Soluție

Pentru funcțiile cu nume, am vrea să fie ca și cum am calcula mereu tipul

$$\vdash \texttt{let}\ id = (\lambda x.x)\ \texttt{in}\ \texttt{if}\ id\ \mathit{true}\ \texttt{then}\ id\ 3\ \texttt{else}\ 4) : \texttt{int}$$

Operațional: redenumim variabilele de tip când instanțiem numele funcției

Scheme de tipuri

- Numim schemă de tipuri o expresie de forma $\langle \tau \rangle$,
  unde τ este e un expresie tip cu variabile
- variabilele dintr-o schemă nu pot fi constrânse
  e ca și cum ar fi cuantificate universal
- O schemă poate fi concretizată la un tip obișnuit substituindu-i fiecare variabilă cu orice tip (poate fi și variabilă)

Reguli pentru scheme

$$(:\text{LET}) \quad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma_1 \vdash e_2 : \tau}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e2 : \tau} \quad \textit{dacă}\ \Gamma_1 = \Gamma[\langle \tau_1 \rangle / x]$$

Reguli pentru scheme

(:LET) $$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma_1 \vdash e_2 : \tau}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e2 : \tau} \quad \textit{dacă}\ \Gamma_1 = \Gamma[\langle\tau_1\rangle/x]$$

```
type(Gamma, let(X, E1, E2), T) :-
    type(Gamma, E1, T1),
    copy_term(T1, FreshT1),   % redenumeste variabilele
                              % ca sa nu poata fi constranse
    set(Gamma, X, scheme(FreshT1), GammaX),
    type(GammaX, E2, T).
```

Reguli pentru scheme

(:SCH) $\Gamma \vdash x : \tau'$ *dacă* $\Gamma(x) = \langle\tau\rangle$ și $\tau' = \theta(\tau)$

```
type(Gamma, X, T) :-
    atom(X), get(Gamma, X, T), is_type(T).
type(Gamma, X, T) :-
    atom(X), get(Gamma, X, scheme(TX)),
    copy_term(T1, T).          % redenumeste variabilele
                               % ca sa poata fi constranse
```

Exemple

Prolog

```
  run(E) :-
    write("Program "),
    write(E),
    type(([],[]), E, T)
    ->  write(" has type "), write(T), nl
    ;   write(" doesn't type").

 ?- run(id -> if(id $ true, id $ 3, 4 )).
Program id->if(id$true,id$3,4) doesn't type
true.

?- run(let(id, x -> x, if(id $ true, id $ 3, 4 ))).
Program let(id,(x->x),if(id$true,id$3,4)) has type int
true
```

Succes la examen!