# Curs 10

2020-2021

Fundamentele limbajelor de programare

# Cuprins


1 Semantica Small-Step pentru Lambda Calcul

2 Determinarea tipurilor
- Asociere de tipuri
- Proprietăți
- Exemplu
- Implementare în Prolog

3 Funcții polimorfice

# Sintaxa limbajului LAMBDA

## BNF

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

## Verificarea sintaxei în Prolog

```
exp(Id) :- atom(Id).                              % identifier
exp(Lit) :- Lit = true ; Lit = false ; integer(Lit).
exp(E1 + E2) :- exp(E1), exp(E2).                 % same for any op
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).         % application
exp(let(Id, Exp1, Exp2)) :- atom(Id), exp(Exp1), exp(Exp2).
```

# Semantica small-step pentru Lambda

- Definește cel mai mic pas de execuție ca o relație de tranziție între expresii dată fiind o stare cu valori pentru variabilele libere

  $\rho \vdash cod \rightarrow cod'$ `step(Env, Cod1,Cod2)`
- Execuția se obține ca o succesiune de astfel de tranziții.

# Semantica variabilelor

$$\rho \vdash x \rightarrow v \quad \textit{dacă}\ \rho(x) = v$$

## Prolog

```
step(Env, X, V) :- atom(X), get(Env, X, V).
```

# Semantica expresiilor aritmetice

- Semantica adunării a două expresii aritmetice

$$\langle i_1 + i_2 \ , \ \sigma \rangle \to \langle i \ , \ \sigma \rangle \quad \textit{dacă } i = i_1 + i_2$$

$$\frac{\langle a_1 \ , \ \sigma \rangle \to \langle a_1' \ , \ \sigma' \rangle}{\langle a_1 + a_2 \ , \ \sigma \rangle \to \langle a_1' + a_2 \ , \ \sigma' \rangle} \qquad \frac{\langle a_2 \ , \ \sigma \rangle \to \langle a_2' \ , \ \sigma' \rangle}{\langle a_1 + a_2 \ , \ \sigma \rangle \to \langle a_1 + a_2' \ , \ \sigma' \rangle}$$

- Pentru alți operatori (artimetici, de comparație, booleeni, condițional)
  - Similar cu regulile din IMP

## Prolog

```
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).
```

# Semantica $\lambda$-abstracției

$$\rho \vdash \lambda x.e \rightarrow \mathit{closure}(x, e, \rho)$$

$\lambda$-abstracția se evaluează la o valoare specială numită closure care capturează valorile curente ale variabilelor pentru a se putea executa în acest mediu atunci când va fi aplicată.

**Prolog**

```
step(Env, X -> E, closure(X, E, Env)).
```

# Semantica construcției `let`

$$\rho \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \rightarrow (\lambda x.e2)\ e_1$$

A îi da lui $x$ valoarea lui $e_1$ în $e_2$ este același lucru cu a aplica funcția de $x$ cu corpul $e_2$ expresiei $e_1$.

## Prolog

```
step(_, let(X, E1, E2), (X -> E2) $ E1).
```

# Semantica operatorului de aplicare

$$\frac{\rho_e[v/x] \vdash e \rightarrow e'}{\rho \vdash closure(x, e, \rho_e)\ v \rightarrow closure(x, e', \rho_e)\ v} \quad \textit{dacă } v \text{ valoare}$$

$$\rho \vdash closure(x, v, \rho_e)\ e \rightarrow v \quad \textit{dacă } v \text{ valoare}$$

$$\frac{\rho \vdash e_1 \rightarrow e_1'}{\rho \vdash e_1\ e_2 \rightarrow e_1'\ e_2} \qquad \frac{\rho \vdash e_2 \rightarrow e_2'}{\rho \vdash e_1\ e_2 \rightarrow e_1\ e_2'}$$

## Prolog

```
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) :-
    \+ step(Env, V, _),
    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**

- $2\ (\lambda x.x)$
- $(\lambda x.x) + 1$
- $(\lambda x.x + 1)\ (\lambda x.x)$

# 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

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

# Relația de asociere de tipuri

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

- $\tau$ este un tip

  $\tau ::= \mathit{int}$ [întregi]
  $\quad\mid \mathit{bool}$ [valori de adevăr]
  $\quad\mid \tau \to \tau$ [funcții]
  $\quad\mid a$ [variabile de tip]

- $e$ este un termen (potențial cu variabile libere)
- $\Gamma$ 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 $\tau$.

# Axiome

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

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

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

# Expresii

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

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

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

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

# Fragmentul funcțional

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

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

# Programe în execuție

**Problemă:**

- În timpul execuției programul conține valori de tip closure
- Care este tipul lor?

## Soluție

- Adăugam regula (:CL) $\dfrac{\Gamma_\rho \vdash \lambda x.e : \tau}{\Gamma \vdash closure(x, e, \rho) : \tau}$ *unde*
- Mediul de tipuri $\Gamma_\rho$ asociat unui mediu de execuție $\rho$ satisface:
  - $Dom(\Gamma_\rho) = Dom(\rho)$
  - Pentru orice variabilă $x \in Dom(\rho)$, există $\tau$ tip și $v$ valoare astfel încât $\Gamma_\rho(x) = \tau$, $\rho(x) = v$ și $\vdash v : \tau$

# Proprietăți

## Theorem (Proprietatea de a progresa)

*Dacă* $\Gamma_\rho \vdash e : \tau$ *atunci* $e$ *este valoare sau* $e$ *poate progresa în* $\rho$*: există* $e'$ *astfel încât* $\rho \vdash e \to e'$.

## Theorem (Proprietatea de conservare a tipului)

*Dacă* $\Gamma_\rho \vdash e : \tau$ *și* $\rho \vdash e \to e'$*, atunci* $\Gamma'_\rho \vdash e' : \tau$.

## Theorem (Siguranță—programele bine formate nu se împotmolesc)

*Dacă* $\Gamma_\rho \vdash e : \tau$ *și* $\rho \vdash e \longrightarrow^* e'$*, atunci* $e'$ *este valoare sau există* $e''$*, astfel încât* $\rho \vdash e' \to e''$.

# Probleme computaționale

## Verificarea tipului

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

## Determinarea (inferarea) tipului

Date fiind $\Gamma$ și $e$, găsiți (sau arătați ce nu există) un $\tau$ 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)

# Probleme computaționale

**Theorem (Determinarea tipului este decidabilă)**

*Date fiind $\Gamma$ și e, poate fi găsit (sau demonstrat că nu există) un $\tau$ astfel încât $\Gamma \vdash e : \tau$.*

**Theorem (Verificarea tipului este decidabilă)**

*Date fiind $\Gamma$, e și $\tau$, problema $\Gamma \vdash e : \tau$ este decidabilă.*

**Theorem (Unicitatea tipului)**

*Dacă $\Gamma \vdash e : \tau$ și $\Gamma \vdash e : \tau'$, atunci $\tau = \tau'$.*

# Exemplu

## Care este tipul expresiei următoare (dacă are)

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

## Aplicăm regula

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

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

# Exemplu

**Care este tipul expresiei următoare (dacă are)**

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

**Aplicăm regula**

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

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

# Exemplu

## Care este tipul expresiei următoare (dacă are)

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

## Aplicăm regula

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

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

# Exemplu

## Unde suntem

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

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

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

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

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y = 0$ :**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

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

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x/y$ :**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

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

## 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 $t_y = \mathtt{int}$
$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash z : t_z$ 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, $t_x = \mathtt{int}$

# Exemplu

**Finalizăm**

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

**Rezolvăm constrângerile și obținem**

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

# 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 $\lambda$-expresie scrisă cu sintaxa descrisă mai sus
- `T` este o expresie de tip cu variabile

# Sintaxa limbajului LAMBDA

## BNF

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

## Verificarea sintaxei în Prolog

```
exp(Id) :- atom(Id).                              % identifier
exp(Lit) :- Lit = true ; Lit = false ; integer(Lit).
exp(E1 + E2) :- exp(E1), exp(E2).                % same for any op
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).        % application
exp(let(Id, Exp1, Exp2)) :- atom(Id), exp(Exp1), exp(Exp2).
```

# Sintaxa tipurilor

## BNF

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

## Verificarea sintaxei tipurilor în Prolog

```
is_type(X) :- variable(X).       % variabile
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

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

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

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

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

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

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

# Expresia condițională

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

```
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 $t$
- $\vdash$ **if** $(\lambda x.x)$ *true* **then** $(\lambda x.x)$ 3 **else** 4 :**int**

# Tipurile variabile nu sunt suficiente

## Tipurile variabile sunt destul de flexibile

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

Dar tipul unei expresii este fixat:

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

# Tipurile variabile nu sunt suficiente

## Tipurile variabile sunt destul de flexibile

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

Dar tipul unei expresii este fixat:

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

## Soluție

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

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

Operațional: redenumim variabilele de tip când instanțiem numele funcției

# Scheme de tipuri

- Numim schemă de tipuri o expresie de forma $\langle\tau\rangle$, unde $\tau$ 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)
  - Pentru orice substituție $\theta$ de la variabile de tip la tipuri cu variabile

# Reguli pentru scheme

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

# Reguli pentru scheme

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

```
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

(:LET) $\dfrac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma_1 \vdash e_2 : \tau}{\Gamma \vdash \textbf{let } x = e_1 \textbf{ in } e2 : \tau}$ *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).
```

(: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(TX, T).          % redenumeste variabilele
                               % ca sa poata fi constranse
```

Pe săptămâna viitoare!