Curs 10

Fundamentele limbajelor de programare

Departamentul de Informatic, FMI, UNIBUC
traian.serbanuta@unibuc.ro

2020-2021

Cuprins

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).                              % 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

- Definete cel mai mic pas de execuie ca o relaie de tranziie între expresii dat fiind o stare cu valori pentru variabilele libere

  $\rho \vdash cod \rightarrow cod'$ `step(Env, Cod1,Cod2)`
- Execuia se obine ca o succesiune de astfel de tranziii.

Semantica variabilelor

$$\rho \vdash x \to v \quad \textit{dac}\, \rho(x) = v$$

Prolog

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

Semantica expresiilor aritmetice

- Semantica adunrii a dou expresii aritmetice

  $\langle i_1 + i_2 \, , \, \sigma \rangle \rightarrow \langle i \, , \, \sigma \rangle \quad dac \; i = i_1 + i_2$

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

- Pentru ali operatori (artimetici, de comparaie, booleeni, condiional)
  - 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 λ-abstraciei

$$\rho \vdash \lambda x.e \to closure(x, e, \rho)$$

λ-abstracia se evalueaz la o valoare special numit closure care captureaz valorile curente ale variabilelor pentru a se putea executa în acest mediu atunci când va fi aplicat.

Prolog

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

Semantica construciei `let`

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

A îi da lui x valoarea lui e_1 în e_2 este acelai lucru cu a aplica funcia de x cu corpul e_2 expresiei e_1.

Prolog

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

Semantica operatorului de aplicare

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

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

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

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.
```

Problem: Sintaxa este prea permisiv

Problem: Muli termeni acceptai de sintax nu pot fi evaluai

- 2 (fun(x)->x)
- (fun(x)->x) $+$ 1
- (fun(x)->$x + 1$) (fun(x)->x)

Problem: Sintaxa este prea permisiv

Problem: Muli termeni acceptai de sintax nu pot fi evaluai

- 2 (fun(x)–>x)— expresia din stânga aplicaiei trebuie s reprezinte o funcie
- (fun(x)–>x) $+$ 1 — adunm funcii cu numere
- (fun(x)–>$x + 1$) (fun(x)–>x) — pot face o reducie, dar tot nu pot evalua

Soluie: Identificarea (precis) a programelor corecte

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

$$((\text{fun}(x) {-}{>} x + 1)\ ((\text{fun}(x) {-}{>} x)\ 3)) : \text{int}$$

Relaia de asociere de tipuri

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

- τ este un tip

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

- e este un termen (potenial cu variabile libere)
- Γ este mediul de tipuri, o funcie parial finit care asociaz tipuri variabilelor (libere ale lui e)
- Variabilele de tip sunt folosite pentru a indica polimorfismul

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

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

Axiome

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

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

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

Expresii

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

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

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

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

Fragmentul funcional

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \mathrm{fun}(x){-}{>}e : \tau \to \tau'} \quad dac\ \Gamma' = \Gamma[x \mapsto \tau]$$

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

Programe în execuie

Problem:

- În timpul execuiei programul conine valori de tip closure
- Care este tipul lor?

Soluie

- Adugam regula (:CL) $\dfrac{\Gamma_\rho \vdash \lambda x.e : \tau}{\Gamma \vdash closure(x, e, \rho) : \tau}$ *unde*
- Mediul de tipuri Γ_ρ asociat unui mediu de execuie ρ satisface:
  - $Dom(\Gamma_\rho) = Dom(\rho)$
  - Pentru orice variabil $x \in Dom(\rho)$, exist τ tip i v valoare astfel încât $\Gamma_\rho(x) = \tau$, $\rho(x) = v$ i $\vdash v : \tau$

Proprieti

ale sistemului de tipuri pentru limbajul fun-IMP

Theorem (Proprietatea de a progresa)

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

Theorem (Proprietatea de conservare a tipului)

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

Theorem (Siguran—programele bine formate nu se împotmolesc)

Dac $\Gamma_\rho \vdash e : \tau$ *i* $\rho \vdash e \longrightarrow^* e'$, *atunci* e' *este valoare sau exist* e'', *astfel încât* $\rho \vdash e' \to e''$.

Probleme computaionale

Verificarea tipului

Date fiind Γ, e i τ, verificai dac $\Gamma \vdash e : \tau$.

Determinarea (inferarea) tipului

Date fiind Γ i e, gsii (sau artai ce nu exist) un τ astfel încât $\Gamma \vdash e : \tau$.

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

Probleme computaionale

Proprieti

Theorem (Determinarea tipului este decidabil)

Date fiind Γ i e, poate fi gsit (sau demonstrat c nu exist) un τ astfel încât $\Gamma \vdash e : \tau$.

Theorem (Verificarea tipului este decidabil)

Date fiind Γ, e i τ, problema $\Gamma \vdash e : \tau$ este decidabil.

Theorem (Unicitatea tipului)

Dac $\Gamma \vdash e : \tau$ i $\Gamma \vdash e : \tau'$, atunci $\tau = \tau'$.

Exemplu

Care este tipul expresiei urmtoare (dac are)

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

Aplicm regula

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \text{fun}(x)\text{–>}e : \tau \to \tau'} \quad \textit{dac}\ \Gamma' = \Gamma[x \mapsto \tau]$$

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

Exemplu

Care este tipul expresiei urmtoare (dac are)

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

Aplicm regula

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \text{fun}(x)\text{->}e : \tau \to \tau'} \quad \textit{dac}\ \Gamma' = \Gamma[x \mapsto \tau]$$

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

Exemplu

Care este tipul expresiei urmtoare (dac are)

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

Aplicm regula

$$(:\text{FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \text{fun}(x)\text{->}e : \tau \to \tau'}\quad dac\ \Gamma' = \Gamma[x \mapsto \tau]$$

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

Exemplu

Unde suntem

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

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

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

Exemplu

Aplicm regula

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

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

Aplicm regula (:INT) $\Gamma \vdash n : int$ *dac n* întreg

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

Aplicm regula

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

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

Exemplu

Recapitulm

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

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y : \texttt{int}$ i $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash z : t_1$

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash x : \texttt{int}$ i $x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash y : \texttt{int}$

i $t_0 = t_z \to t_1$, $t = t_y \to t_0$, $t_1 = \texttt{int}$.

Aplicm regula (:VAR) $\Gamma \vdash x : \tau$ *dac* $\Gamma(x) = \tau$

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

$x \mapsto t_x, y \mapsto t_y, z \mapsto t_z \vdash z : t_z$ adevrat i, de mai sus, $t_1 = t_z$

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

Exemplu

Finalizm

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

Rezolvm constrângerile i obinem

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

Relaia de asociere de tipuri în Prolog

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

- `Gamma` este o list de perechi de forma `(X, T)` unde `X` este un identificator i `T` este o expresie de tip cu variabile
- `E` este o λ-expresie scris cu sintaxa descris mai sus
- `T` este o expresie de tip cu variabile

Sintaxa limbajului LAMBDA

BNF

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

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

τ ::= *int* [întregi]
  | *bool* [valori de adevr]
  | $\tau \to \tau$ [funcii]
  | *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

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

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

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

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

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

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

Expresii

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

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

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

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

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

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

Expresia condiional

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

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

Fragmentul funcțional

$$(\text{:FN})\quad \frac{\Gamma' \vdash e : \tau'}{\Gamma \vdash \mathsf{fun}(x)\text{->}e : \tau \to \tau'} \quad dac\ \Gamma' = \Gamma[x \mapsto \tau]$$

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

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:

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

Soluie

Pentru funciile cu nume, am vrea s fie ca i cum am calcula mereu tipul

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

Operaional: redenumim variabilele de tip când instaniem numele funciei

Scheme de tipuri

- Numim schem de tipuri o expresie de forma $\langle \tau \rangle$, unde τ este e un expresie tip cu variabile
- variabilele dintr-o schem nu pot fi constrânse e ca i cum ar fi cuantificate universal
- O schem poate fi concretizat la un tip obinuit substituindu-i fiecare variabil cu orice tip (poate fi i variabil)
  - Pentru orice substituie θ de la variabile de tip la tipuri cu variabile

Reguli pentru scheme

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

Reguli pentru scheme

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

```
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) $\dfrac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma_1 \vdash e_2 : \tau}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e2 : \tau}$ $\quad$ *dac* $\Gamma_1 = \Gamma[\langle \tau_1 \rangle / x]$

```
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) $\Gamma \vdash x : \tau'$ $\quad$ *dac* $\Gamma(x) = \langle \tau \rangle$ i $\tau' = \theta(\tau)$

```
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
```

Pe sptmâna viitoare!