Curs 13

2021-2022

Fundamentele limbajelor de programare

Semantica small-step pentru λ-calcul

Sintaxa limbajului LAMBDA

BNF

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

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

- 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)`

- Mediul de evaluare ρ este format din perechi variabilă-valoare (x, v) unde variabilele sunt reprezentate prin identificatori, iar valorile sunt *întregi*, *booleene*, sau *valori funcție*.

- Execuția se obține ca o succesiune de astfel de tranziții.

Semantica variabilelor

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

Prolog

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

Semantica expresiilor aritmetice

- Semantica adunării a două expresii aritmetice

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

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

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

Semantica expresiilor aritmetice

- Semantica adunării a două expresii aritmetice

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

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

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-abstracției

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

λ-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`

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

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

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

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

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

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

Semantica operatorului de aplicare

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

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

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

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) :-
      set(EnvE, X, V, EnvEX),
      step(EnvEX, E, E1)  ->
              Result = closure(X, E1, EnvE) $ V
              ;  ( Result = E).
```

Semantica small-step: toți pașii

Prolog

```
all_steps(Env, E1, Trace, E) :-
    step(Env, E1, E2)
    ->  all_steps(Env, E2, Trace1, E), Trace = [E2|Trace1]
    ;   E = E1, Trace=[].

run(E, V) :- all_steps([], E, _, V).
```

Exemplu

```
lam3((x -> ((y -> (x + y)) $ 7)) $ 3).

?- lam3(X), run(X,V).
X =  (x->(y->x+y)$7)$3,
V = 10
```

Semantica small-step: toți pașii

Prolog

```
all_steps(Env, E1, Trace, E) :-
    step(Env, E1, E2)
    ->  all_steps(Env, E2, Trace1, E), Trace = [E2|Trace1]
    ;   E = E1, Trace=[].

print_list([]).
print_list([H|T]) :- print(H), nl, print_list(T).

trace(E1) :- all_steps([], E1, Trace, _), print_list(Trace).
```

Semantica small-step: toți pașii

Exemplu

```
lam3((x -> ((y -> (x + y)) $ 7)) $ 3).


?- lam3(X), trace(X).
closure(x,(y->x+y)$7,[])$3
closure(x,closure(y,x+y,[(x,3)])$7,[])$3
closure(x,closure(y,3+y,[(x,3)])$7,[])$3
closure(x,closure(y,3+7,[(x,3)])$7,[])$3
closure(x,closure(y,10,[(x,3)])$7,[])$3
closure(x,10,[])$3
10
X =  (x->(y->x+y)$7)$3.
```

Pe săptămâna viitoare!