Curs 9

2020-2021

Fundamentele limbajelor de programare

Cuprins

Limbajul IMP

Limbajul IMP

Vom implementa un limbaj care conține:

- Expresii
  - Aritmetice — `x + 3`
  - Booleene — `x >= 7`
- Instrucțiuni
  - De atribuire — `x = 5`
  - Condiționale — `if(x >= 7, x =5, x = 0)`
  - De ciclare — `while(x >= 7,x = x - 1)`
- Compunerea instrțiunilor — `x=7;while(x>=0,x=x-1)`
- Blocuri de instrucțiuni — `{x=7;while(x>=0,x=x-1)}`

Limbajul IMP

Exemplu

Un program în limbajul IMP

```
{x = 10 ; sum = 0;
while(0 =< x,
      {sum = sum + x; x = x-1}
      )},sum
```

- Semantica
  după execuția programului, se evaluează `sum`

Sintaxa BNF a limbajului IMP

$$
\begin{aligned}
E &::= n \mid x \\
&\quad \mid E + E \mid E - E \mid E * E \\
\\
B &::= \texttt{true} \mid \texttt{false} \\
&\quad \mid E =< E \mid E >= E \mid E == E \\
&\quad \mid \texttt{not}(B) \mid \texttt{and}(B , B) \mid \texttt{or}(B , B) \\
\\
C &::= \texttt{skip} \\
&\quad \mid x = E \\
&\quad \mid \texttt{if}(B , C , C) \\
&\quad \mid \texttt{while}(B , C) \\
&\quad \mid \{ C \} \mid C ; C \\
\\
P &::= \{ C \}, E
\end{aligned}
$$

O implementare a limbajului IMP în Prolog

Decizii de implementare

- {} și ; sunt operatori

```
:- op(100, xf, {}).
:- op(1100, yf, ;).
```

- definim un predicat pentru fiecare categorie sintactică

```
stmt(while(BE,St)) :- bexp(BE),  stmt(St).
```

- while, if, and, etc sunt functori în Prolog
  `while(true,skip)` este un termen compus

- , are semnificația obișnuită

- pentru valori numerice folosim întregii din Prolog

```
aexp(I) :- integer(I).
```

- pentru identificatori folosim atomii din Prolog

```
aexp(X) :- atom(X).
```

Expresiile aritmetice

$$E ::= n \mid x \mid E + E \mid E - E \mid E * E$$

Prolog

```
aexp(I) :- integer(I).
aexp(X) :- atom(X).
aexp(A1 + A2) :- aexp(A1), aexp(A2).
aexp(A1 - A2) :- aexp(A1), aexp(A2).
aexp(A1 * A2) :- aexp(A1), aexp(A2).
```

Expresiile aritmetice

Exemplu

```
?- aexp(1000).
true.
?- aexp(id).
true.
?- aexp(id + 1000).
true.
?- aexp(2 + 1000).
true.
?- aexp(x * y).
true.
?- aexp(- x).
false.
```

Expresiile booleene

$B ::=$ **true** | **false**
| E =< E | E >= E | E == E
| not(B) | and(B , B) | or(B , B)

Prolog

```
bexp(true). bexp(false).
bexp(and(BE1,BE2)) :- bexp(BE1), bexp(BE2).
bexp(or(BE1,BE2)) :-  bexp(BE1), bexp(BE2).
bexp(not(BE)) :-  bexp(BE).

bexp(A1 =< A2) :- aexp(A1), aexp(A2).
bexp(A1 >= A2) :- aexp(A1), aexp(A2).
bexp(A1 == A2) :- aexp(A1), aexp(A2).
```

Expresiile booleene

Exemplu

```
?- bexp(true).
true.
?- bexp(id).
false.
?- bexp(not(1 =< 2)).
true.
?- bexp(or(1 =< 2,true)).
true.
?- bexp(or(a =< b,true)).
true.
?- bexp(not(a)).
false.
?- bexp(!(a)).
false.
```

Instrucțiunile

$C ::=$ **skip**
| x = E ;
| **if(** B **)** C **else** C
| **while(** B **)** C
| **{** C **}** | C ; C

Prolog

```
stmt(skip).
stmt(X = AE) :- atom(X), aexp(AE).
stmt(St1;St2) :- stmt(St1), stmt(St2).
stmt((St1;St2)) :- stmt(St1), stmt(St2).
stmt({St}) :- stmt(St).
stmt(if(BE,St1,St2)) :-  bexp(BE), stmt(St1), stmt(St2).
stmt(while(BE,St)) :- bexp(BE), stmt(St).
```

Instrucțiunile

Exemplu

```
?- stmt(id = 5).
true.
?- stmt(id = a).
true.
?- stmt(3 = 6).
false.
?- stmt(if(true, x=2;y=3, x=1;y=0)).
true.
?- stmt(while(x =< 0,skip)).
true.
?- stmt(while(x =< 0,)).
false.
?- stmt(while(x =< 0,skip)).
true .
```

Programele

$P ::= \{\ C\ \},\ E$

Prolog

```
program(St,AE) :- stmt(St), aexp(AE).
```

Exemplu

```
test0 :- program( {x = 10 ; sum = 0;
                   while(0 =< x,
                         {sum = sum + x; x = x-1}
                        )}
                   , sum).
```

?- test0.
true.

O implementare a semanticii small-step

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
  - Semantică Operațională Structurală
  - semantică prin tranziții
  - semantică prin reducere
- Definește cel mai mic pas de execuție ca o relație „de tranziție” între configurații:

$$\langle cod\ ,\ \sigma \rangle \rightarrow \langle cod\ ,\ \sigma' \rangle$$

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
  - Semantică Operațională Structurală
  - semantică prin tranziții
  - semantică prin reducere
- Definește cel mai mic pas de execuție ca o relație „de tranziție” între configurații:

$$\langle cod\ ,\ \sigma \rangle \to \langle cod\ ,\ \sigma' \rangle$$

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

$$\langle \texttt{int}\ x = 0\ \texttt{;}\ x = x + 1\ \texttt{;}\ ,\ \bot \rangle \quad \longrightarrow \quad \langle x = x + 1\ \texttt{;}\ ,\ x \mapsto 0 \rangle$$

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
  - Semantică Operațională Structurală
  - semantică prin tranziții
  - semantică prin reducere
- Definește cel mai mic pas de execuție ca o relație „de tranziție” între configurații:

$$\langle cod\ ,\ \sigma \rangle \rightarrow \langle cod\ ,\ \sigma' \rangle$$

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

$$\begin{aligned} \langle \texttt{int}\ x = 0\ \texttt{;}\ x = x + 1\ \texttt{;}\ ,\ \bot \rangle &\longrightarrow \langle x = x + 1\ \texttt{;}\ ,\ x \mapsto 0 \rangle \\ &\longrightarrow \langle x = 0 + 1\ \texttt{;}\ ,\ x \mapsto 0 \rangle \end{aligned}$$

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
  - Semantică Operațională Structurală
  - semantică prin tranziții
  - semantică prin reducere
- Definește cel mai mic pas de execuție ca o relație „de tranziție” între configurații:

$$\langle cod\ ,\ \sigma\rangle \to \langle cod\ ,\ \sigma'\rangle$$

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

$$\begin{aligned}\langle \texttt{int}\ x = 0\ \texttt{;}\ x = x + 1\ \texttt{;}\ ,\ \bot\rangle &\longrightarrow \langle x = x + 1\ \texttt{;}\ ,\ x \mapsto 0\rangle \\ &\longrightarrow \langle x = 0 + 1\ \texttt{;}\ ,\ x \mapsto 0\rangle \\ &\longrightarrow \langle x = 1\ \texttt{;}\ ,\ x \mapsto 0\rangle\end{aligned}$$

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
  - Semantică Operațională Structurală
  - semantică prin tranziții
  - semantică prin reducere
- Definește cel mai mic pas de execuție ca o relație „de tranziție” între configurații:

$$\langle cod\ ,\ \sigma\rangle \to \langle cod\ ,\ \sigma'\rangle$$

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

$$\begin{aligned}
\langle \texttt{int}\ x = 0\ \texttt{;}\ x = x + 1\ \texttt{;}\ ,\ \bot\rangle &\longrightarrow \langle x = x + 1\ \texttt{;}\ ,\ x \mapsto 0\rangle \\
&\longrightarrow \langle x = 0 + 1\ \texttt{;}\ ,\ x \mapsto 0\rangle \\
&\longrightarrow \langle x = 1\ \texttt{;}\ ,\ x \mapsto 0\rangle \\
&\longrightarrow \langle \texttt{\{\}}\ ,\ x \mapsto 1\rangle
\end{aligned}$$

Semantica small-step

- Definește cel mai mic pas de execuție ca o relație de tranziție între configurații:

  $\langle cod\ ,\ \sigma\rangle \rightarrow \langle cod'\ ,\ \sigma'\rangle$ `step(Cod,S1,Cod',S2)`
- Execuția se obține ca o succesiune de astfel de tranziții.
- Starea executiei unui program IMP la un moment dat este o funcție parțială: $\sigma = n \mapsto 10, sum \mapsto 0$, etc.

Reprezentarea stărilor în Prolog

```
get(S,X,I) :- member(vi(X,I),S).
get(_,_,0).
set(S,X,I,[vi(X,I)|S1]) :- del(S,X,S1).

del(S,X,S1) :- select(vi(X,_), S, S1), !.
del(S, _, S).
```

Semantica expresiilor aritmetice

- Semantica unei variabile

  $\langle x\ ,\ \sigma\rangle \to \langle i\ ,\ \sigma\rangle \quad$ *dacă* $i = \sigma(x)$

Prolog

```
step(X,S,I,S) :-
  atom(X),
  get(S,X,I).
```

Semantica expresiilor aritmetice

- Semantica adunării a două expresii aritmetice

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

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

Prolog

```
step(I1 + I2,S,I,S):- integer(I1),integer(I2),
                              I is I1 + I2.

step(AE + AE1,S1,AE + AE2,S1):- step(AE1,S1,AE2,S2).

step(AE1 + AE,S1,AE2 + AE,S2):- step(AE1,S1,AE2,S2).
```

Semantica expresiilor aritmetice

Exemplu

?- step(a + b, [vi(a,1),vi(b,2)],AE, S).
AE = 1+b,
S = [vi(a, 1), vi(b, 2)] .

?- step(1 + b, [vi(a,1),vi(b,2)],AE, S).
AE = 1+2,
S = [vi(a, 1), vi(b, 2)] .

?- step(1 + 2, [vi(a,1),vi(b,2)],AE, S).
AE = 3,
S = [vi(a, 1), vi(b, 2)]

Semantica expresiilor aritmetice

Exemplu

?- step(a + b, [vi(a,1),vi(b,2)],AE, S).
AE = 1+b,
S = [vi(a, 1), vi(b, 2)] .

?- step(1 + b, [vi(a,1),vi(b,2)],AE, S).
AE = 1+2,
S = [vi(a, 1), vi(b, 2)] .

?- step(1 + 2, [vi(a,1),vi(b,2)],AE, S).
AE = 3,
S = [vi(a, 1), vi(b, 2)]

- Semantica * și - se definesc similar.

Semantica expresiilor booleene

- Semantica operatorului de comparație

$\langle i_1 =< i_2 \ , \ \sigma \rangle \to \langle \mathbf{false} \ , \ \sigma \rangle \quad$ *dacă* $i_1 > i_2$

$\langle i_1 =< i_2 \ , \ \sigma \rangle \to \langle \mathbf{true} \ , \ \sigma \rangle \quad$ *dacă* $i_1 \leq i_2$

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

Prolog

```
step(I1 =< I2,S,true,S):- integer(I1),integer(I2),
                                    (I1 =< I2).
step(I1 =< I2,S,false,S):- integer(I1),integer(I2),
                                     (I1 > I2).
step(AE =< AE1,S1,AE =< AE2,S2):- step(AE1,S1,AE2,S2).
step(AE1 =< AE,S1,AE2 =< AE,S2):- step(AE1,S1,AE2,S2).
```

Semantica expresiilor Booleene

- Semantica negației

$$\langle \textbf{not(true)} \ , \ \sigma \rangle \rightarrow \langle \textbf{false} \ , \ \sigma \rangle$$

$$\langle \textbf{not(false)} \ , \ \sigma \rangle \rightarrow \langle \textbf{true} \ , \ \sigma \rangle$$

$$\frac{\langle a \ , \ \sigma \rangle \rightarrow \langle a' \ , \ \sigma' \rangle}{\langle \textbf{not} \ (a) \ , \ \sigma \rangle \rightarrow \langle \textbf{not} \ (a') \ , \ \sigma' \rangle}$$

Prolog

```
step(not(true),S,false,S) .
step(not(false),S,true,S) .

step(not(BE1),S1,not(BE2),S2) :- step(BE1,S1,BE2,S2).
```

Semantica compunerii și a blocurilor

- Semantica blocurilor

  $\langle \{\ s\ \}\ ,\ \sigma \rangle \to \langle s\ ,\ \sigma \rangle$

- Semantica compunerii secvențiale

  $\langle \{\}; s_2\ ,\ \sigma \rangle \to \langle s_2\ ,\ \sigma \rangle \qquad \dfrac{\langle s_1\ ,\ \sigma \rangle \to \langle s_1'\ ,\ \sigma' \rangle}{\langle s_1; s_2\ ,\ \sigma \rangle \to \langle s_1'; s_2\ ,\ \sigma' \rangle}$

Prolog

```
step({E},S,E,S).

step((skip;St2),S,St2,S).
step((St1;St),S1,(St2;St),S2) :-
                                step(St1,S1,St2,S2) .
```

Semantica atribuirii

Semantica atribuirii

$\langle x = i\ ,\ \sigma\rangle \rightarrow \langle \{\}\ ,\ \sigma'\rangle \quad \textit{dacă}\ \sigma' = \sigma[i/x]$

$$\frac{\langle a\ ,\ \sigma\rangle \rightarrow \langle a'\ ,\ \sigma'\rangle}{\langle x = a\ ,\ \sigma\rangle \rightarrow \langle x = a'\ ;\ ,\ \sigma'\rangle}$$

Prolog

```
step(X = I,S,skip,S1) :- integer(I),set(S,X,I,S1).

step(X = AE1,S1,X = AE2,S2) :-
                                    step(AE1,S1,AE2,S2).
```

Semantica lui `if`

Semantica lui `if`

$$\langle \mathbf{if}\ (\mathbf{true}, bl_1, bl_2)\ ,\ \sigma\rangle \to \langle bl_1\ ,\ \sigma\rangle$$

$$\langle \mathbf{if}\ (\mathbf{false}, bl_1, bl_2)\ ,\ \sigma\rangle \to \langle bl_2\ ,\ \sigma\rangle$$

$$\frac{\langle b\ ,\ \sigma\rangle \to \langle b'\ ,\ \sigma'\rangle}{\langle \mathbf{if}\ (b, bl_1, bl_2)\ ,\ \sigma\rangle \to \langle \mathbf{if}\ (b', bl_1, bl_2)\ ,\ \sigma'\rangle}$$

Prolog

```
step(if(true,St1,_),S,St1,S).
step(if(false,_,St2),S,St2,S).

step(if(BE1,St1,St2),S1,if(BE2,St1,St2),S2) :-
                                    step(BE1,S1,BE2,S2) .
```

Semantica lui `while`

- Semantica lui `while`

$$\langle \textbf{while}\ (b, bl)\ ,\ \sigma\rangle \rightarrow \langle \textbf{if}\ (b, bl\ \textbf{;while}\ (b, bl), \textbf{skip})\ ,\ \sigma\rangle$$

Prolog

```
step(while(BE,St),S,if(BE,(St;while(BE,St)),skip),S).
```

Semantica programelor

- Semantica programelor

$$\frac{\langle a_1 \ , \ \sigma_1 \rangle \rightarrow \langle a_2 \ , \ \sigma_2 \rangle}{\langle (\mathbf{skip}, a_1) \ , \ \sigma_1 \rangle \rightarrow \langle (\mathbf{skip}, a_2) \ , \ \sigma_2 \rangle}$$

$$\frac{\langle s_1 \ , \ \sigma_1 \rangle \rightarrow \langle s_2 \ , \ \sigma_2 \rangle}{\langle (s_1, a) \ , \ \sigma_1 \rangle \rightarrow \langle (s_2, a) \ , \ \sigma_2 \rangle}$$

Prolog

```
step((skip,AE1),S1,(skip,AE2),S2) :-
                                step(AE1,S1,AE2,S2) .
step((St1,AE),S1,(St2,AE),S2) :-
                                step(St1,S1,St2,S2).
```

Execuția programelor

Prolog

```
all_steps(P1, S1, PF, SF) :-
    step(P1, S1, P2, S2)
    ->  all_steps(P2, S2, PF, SF)
    ;   PF = P1, SF = S1.

run_program(Name) :- defpg(Name,{P},E),
                     all_steps((P,E), [],(skip, I), _),
                     write(I).
```

Exemplu

```
defpg(pg2, {x = 10 ; sum = 0; while(0 =< x, {
                                  sum = sum + x;
                                  x = x - 1})},sum)
```

```
?- run_program(pg2).
55
true
```

Execuția programelor: `trace`

Putem defini o funcție care ne permite să urmărim execuția unui program în implementarea noastră?

Execuția programelor: `trace`

Putem defini o funcție care ne permite să urmărim execuția unui program în implementarea noastră?

Prolog

```
all_steps(P1, S1, PF, SF, [(P1, S1)| Trace]) :-
    step(P1, S1, P2, S2)
    ->  all_steps(P2, S2, PF, SF, Trace)
    ;   PF = P1, SF = S1, Trace=[].

trace_program(Name) :- defpg(Name,{P},E),
                       all_steps((P,E), [],_, _, Trace),
                       write(Trace).
```

Execuția programelor: `trace_program`

Exemplu

```
?- trace_program(pg2).
. . .
((if(0=<x,(sum=sum+x;x=x-1;while(0=<x,sum=sum+x;x=x-1)),skip), sum),
[vi(x,-1),vi(sum,55)]),
((if(0=<-1,(sum=sum+x;x=x-1;while(0=<x,sum=sum+x;x=x-1)),skip), sum),
[vi(x,-1),vi(sum,55)]),
((if(false,(sum=sum+x;x=x-1;while(0=<x,sum=sum+x;x=x-1)),skip), sum),
[vi(x,-1),vi(sum,55)]),
((skip, sum), [vi(x,-1),vi(sum,55)]),
((skip, 55), [vi(x,-1),vi(sum,55)]),
```

Sintaxa limbajului LAMBDA

BNF

$$
\begin{array}{rcl}
e & ::= & x \\
 & | & \lambda x.e \\
 & | & e\ e \\
 & | & \texttt{let}\ x = e\ \texttt{in}\ e
\end{array}
$$

Verificarea sintaxei în Prolog

```
exp(Id) :- atom(Id).                          % identifier
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

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

Prolog

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

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

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

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

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

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.
```

Pe săptămâna viitoare!