

# Curs 11

# Cuprins

- 1 Limbajul IMP
- 2 O implementare a limbajului IMP în Prolog
- 3 O implementare a semanticii small-step

Acest material urmează cursul introductiv:

T. Șerbănuță, *Semantica Limbajelor de Programare*, master, anul I.

## Limbajul IMP

# Limbajul IMP

Vom implementa un limbaj care conține:

- Expresii

- Aritmetice
- Booleene

```
x + 3  
x >= 7
```

- Instrucțiuni

- De atribuire
- Condiționale
- De ciclare

```
x = 5  
if(x >= 7, x =5, x = 0)  
while(x >= 7,x = x - 1)
```

- Compunerea instrucțiunilor

- Blocuri de instrucțiuni

```
x=7;while(x>=0,x=x-1)  
{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

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

$B ::= \text{true} \mid \text{false}$   
 $\mid E < E \mid E >= E \mid E == E$   
 $\mid \text{not}(B) \mid \text{and}(B, B) \mid \text{or}(B, B)$

$St ::= \text{skip}$   
 $\mid x = E$   
 $\mid \text{if}(B, St, St)$   
 $\mid \text{while}(B, St)$   
 $\mid \{ St \} \mid St ; St$

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

## 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

$B ::= \text{true} \mid \text{false}$   
 $\mid E < E \mid E > E \mid E == E$   
 $\mid \text{not}(B) \mid \text{and}(B, B) \mid \text{or}(B, B)$

## 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

```
St ::= skip  
      | x = E ;  
      | if( B ) St else St  
      | while( B ) St  
      | { St } | St ; St
```

## 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

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

## 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:

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

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

$$\begin{aligned} \langle \text{int } x = 0 ; x = x + 1 ; , \perp \rangle &\rightarrow \langle x = x + 1 ; , x \mapsto 0 \rangle \\ &\rightarrow \langle x = 0 + 1 ; , x \mapsto 0 \rangle \\ &\rightarrow \langle x = 1 ; , x \mapsto 0 \rangle \\ &\rightarrow \langle \{\} , x \mapsto 1 \rangle \end{aligned}$$

## Semantica small-step

- Definește cel mai mic pas de execuție ca o relație de tranziție între configurații:  
 $\langle cod, \sigma \rangle \rightarrow \langle cod', \sigma' \rangle$  smallstep(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([vi(X,_)|S],X,S).  
del([H|S],X,[H|S1]) :- del(S,X,S1).  
del([],_,[]).
```

# Semantica expresiilor aritmetice

## □ Semantica unei variabile

$\langle x, \sigma \rangle \rightarrow \langle i, \sigma \rangle$  dacă  $i = \sigma(x)$

## Prolog

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

# Semantica expresiilor aritmetice

## □ Semantica adunării a două expresii aritmetice

$\langle i_1 + i_2, \sigma \rangle \rightarrow \langle i, \sigma \rangle$  dacă  $i = i_1 + i_2$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle} \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a_1 + a'_2, \sigma \rangle}$$

## Prolog

```
smallstepA(I1 + I2,S,I,S):- integer(I1),integer(I2),  
                             I is I1 + I2.  
  
smallstepA(I + AE1,S,I + AE2,S):- integer(I),  
                                   smallstepA(AE1,S,AE2,S).  
  
smallstepA(AE1 + AE,S,AE2 + AE,S):-  
                                   smallstepA(AE1,S,AE2,S).
```

# Semantica expresiilor aritmetice

## Exemplu

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

AE =  $1+b$ ,

S =  $[vi(a, 1), vi(b, 2)]$  .

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

AE =  $1+2$ ,

S =  $[vi(a, 1), vi(b, 2)]$  .

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

AE =  $3$ ,

S =  $[vi(a, 1), vi(b, 2)]$

□ Semantica  $*$  și  $-$  se definesc similar.

# Semantica expresiilor booleene

## □ Semantica operatorului de comparație

$\langle i_1 =< i_2, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$  dacă  $i_1 > i_2$

$\langle i_1 =< i_2, \sigma \rangle \rightarrow \langle \text{true}, \sigma \rangle$  dacă  $i_1 \leq i_2$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 =< a_2, \sigma \rangle \rightarrow \langle a'_1 =< a_2, \sigma \rangle} \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle a_1 =< a_2, \sigma \rangle \rightarrow \langle a_1 =< a'_2, \sigma \rangle}$$

## Prolog

```
smallstepB(I1 =< I2,S,true,S):- integer(I1),integer(I2),  
                                (I1 =< I2).  
smallstepB(I1 =< I2,S,false,S):- integer(I1),integer(I2),  
                                (I1 > I2).  
smallstepB(I =< AE1,S,I =< AE2,S):- integer(I),  
                                smallstepA(AE1,S,AE2,S).  
smallstepB(AE1 =< AE,S,AE2 =< AE,S):-  
                                smallstepA(AE1,S,AE2,S).
```

# Semantica expresiilor Booleene

## □ Semantica negației

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

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

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

## Prolog

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

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

```
smallstepB(not(BE1),S,not(BE2),S) :-  
    smallstepB(BE1,S,BE2,S) .
```

# Semantica compunerii și a blocurilor

## □ Semantica blocurilor

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

## □ Semantica compunerii secvențiale

$$\langle \{ \} s_2 , \sigma \rangle \rightarrow \langle s_2 , \sigma \rangle \quad \frac{\langle s_1 , \sigma \rangle \rightarrow \langle s'_1 , \sigma' \rangle}{\langle s_1 s_2 , \sigma \rangle \rightarrow \langle s'_1 s_2 , \sigma' \rangle}$$

## Prolog

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

```
smallstepS((skip;St2),S,St2,S).
```

```
smallstepS((St1;St),S1,(St2;St),S2) :-
```

```
    smallstepS(St1,S1,St2,S2) .
```



# Semantica atribuirii

## □ Semantica atribuirii

$\langle x = i, \sigma \rangle \rightarrow \langle \{\} , \sigma' \rangle$  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

```
smallstepS(X = AE,S,skip,S1) :- integer(AE),set(S,X,AE,S1).
```

```
smallstepS(X = AE1,S,X = AE2,S) :-  
    smallstepA(AE1,S,AE2,S).
```

# Semantica lui if

## □ Semantica lui if

$$\langle \text{if } (\text{true}, b_1, b_2) , \sigma \rangle \rightarrow \langle b_1 , \sigma \rangle$$
$$\langle \text{if } (\text{false}, b_1, b_2) , \sigma \rangle \rightarrow \langle b_2 , \sigma \rangle$$
$$\frac{\langle b , \sigma \rangle \rightarrow \langle b' , \sigma \rangle}{\langle \text{if } (b, b_1, b_2) , \sigma \rangle \rightarrow \langle \text{if } (b', b_1, b_2) , \sigma \rangle}$$

## Prolog

```
smallstepS(if(true,St1,_),S,St1,S).
```

```
smallstepS(if(false,_,St2),S,St2,S).
```

```
smallstepS(if(BE1,St1,St2),S,if(BE2,St1,St2),S) :-  
    smallstepB(BE1,S,BE2,S) .
```

# Semantica lui while

## □ Semantica lui while

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

## Prolog

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

# Semantica programelor

## □ Semantica programelor

$$\frac{\langle a_1, \sigma_1 \rangle \rightarrow \langle a_2, \sigma_2 \rangle}{\langle (\text{skip}, a_1), \sigma_1 \rangle \rightarrow \langle (\text{skip}, a_2), \sigma_2 \rangle}$$
$$\frac{\langle s_1, \sigma_1 \rangle \rightarrow \langle s_2, \sigma_2 \rangle}{\langle (s_1, a), \sigma_1 \rangle \rightarrow \langle (s_2, a), \sigma_2 \rangle}$$

## Prolog

```
smallstepP(skip, AE1, S1, skip, AE2, S2) :-  
    smallstepA(AE1, S1, AE2, S2) .  
smallstepP(St1, AE, S1, St2, AE, S2) :-  
    smallstepS(St1, S1, St2, S2) .
```

# Execuția programelor

## Prolog

```
run(skip,I,_,I):- integer(I).  
run(St1,AE1,S1,I) :- smallstepP(St1,AE1,S1,St2,AE2,S2),  
                      run(St2,AE2,S2,I).  
  
run_program(Name) :- defpg(Name,{P},E), run(P,E, [],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?

### Prolog

```
mytrace(skip,I,_) :- integer(I).  
mytrace(St1,AE1,S1) :- smallstepP(St1,AE1,S1,St2,AE2,S2),  
                        write(St2),nl,  
                        write(AE2),nl,  
                        write(S2),nl,  
                        mytrace(St2,AE2,S2).  
  
trace_program(Name) :- defpg(Name,{P},E),  
                        mytrace(P,E,[]).
```

# Execuția programelor: trace\_program

## Exemplu

?- trace\_program(pg2).

...

[vi(x,-1),vi(sum,55)]

if( $0 \leq x$ , (sum=sum+x;x=x-1;while( $0 \leq x$ ,sum=sum+x;x=x-1)),skip)

sum

[vi(x,-1),vi(sum,55)]

if( $0 \leq -1$ , (sum=sum+x;x=x-1;while( $0 \leq x$ ,sum=sum+x;x=x-1)),skip)

sum

[vi(x,-1),vi(sum,55)]

if(false, (sum=sum+x;x=x-1;while( $0 \leq x$ ,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)]

true .



Pe săptămâna viitoare!