

Curs 8-9

Cuprins

- 1 Limbajul IMP
- 2 O implementare a limbajului IMP în Prolog
- 3 Semantica programelor - idei generale
- 4 Semantica small-step
- 5 O implementare a semanticii small-step

Limbajul IMP

Limbajul IMP

Vom implementa un limbaj care conține:

- Expresii

- Aritmetice

$x + 3$

- Booleene

$x \geq 7$

- Instrucțiuni

- De atribuire

$x = 5$

- Conditionale

`if(x >= 7, x = 5, x = 0)`

- De ciclare

`while(x >= 7, x = x - 1)`

- Compunerea instrucțiunilor

`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 executia programului, se evaluează sum

Sintaxa BNF a limbajului IMP

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

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

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

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

O implementare a limbajului IMP în Prolog

Decizii de implementare

- `{}` si `;` 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 semnificatia obisnuită
- 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).
```

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

```
 $B ::= \text{true} \mid \text{false}$   
 $\mid E \leq E \mid E \geq E \mid E == E$   
 $\mid \text{not}(B) \mid \text{and}(B, B) \mid \text{or}(B, B)$ 
```

Prolog

```
bexp(true). bexp(false).  
bexp(and(BE1, BE2)) :- bexp(BE1), bexp(BE2).  
  
bexp(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.

Instructiunile

```
C ::= skip  
    | x = E  
    | if( B , C , 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(if(BE,St1,St2)) :- bexp(BE), stmt(St1), stmt(St2).
```

Instructiunile

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.
```

Semantica programelor - idei generale

Ce înseamnă semantica formală?

Ce definește un limbaj de programare?

Ce înseamnă semantica formală?

Ce definește un limbaj de programare?

- **Sintaxa** – Simboluri de operație, cuvinte cheie, descriere (formală) a programelor/expresiilor bine formate

Ce înseamnă semantica formală?

Ce definește un limbaj de programare?

- **Sintaxa** – Simboluri de operație, cuvinte cheie, descriere (formală) a programelor/expresiilor bine formate
- **Practic** – Un limbaj e definit de modul cum poate fi folosit
 - Manual de utilizare și exemple de bune practici
 - Implementare (compilator/interpretor)
 - Instrumente ajutătoare (analizor de sintaxă, depanator)

Ce înseamnă semantica formală?

Ce definește un limbaj de programare?

- **Sintaxa** – Simboluri de operație, cuvinte cheie, descriere (formală) a programelor/expresiilor bine formate
- **Practic** – Un limbaj e definit de modul cum poate fi folosit
 - Manual de utilizare și exemple de bune practici
 - Implementare (compilator/interpretor)
 - Instrumente ajutătoare (analizor de sintaxă, depanator)
- **Semantica** – Ce înseamnă/care e comportamentul unei instrucțiuni?

La ce folosește semantica?

- Să înțelegem un limbaj în profunzime
 - Ca programator: pe ce mă pot baza când programez în limbajul dat
 - Ca implementator al limbajului: ce garanții trebuie să ofer

La ce folosește semantica?

- Să înțelegem un limbaj în profunzime
 - Ca programator: pe ce mă pot baza când programez în limbajul dat
 - Ca implementator al limbajului: ce garanții trebuie să ofer
- Ca instrument în proiectarea unui nou limbaj/a unei extensii
 - Înțelegerea componentelor și a relațiilor dintre ele
 - Exprimarea (și motivarea) deciziilor de proiectare
 - Demonstrarea unor proprietăți generice ale limbajului

La ce folosește semantica?

- Să înțelegem un limbaj în profunzime
 - Ca programator: pe ce mă pot baza când programez în limbajul dat
 - Ca implementator al limbajului: ce garanții trebuie să ofer
- Ca instrument în proiectarea unui nou limbaj/a unei extensii
 - Înțelegerea componentelor și a relațiilor dintre ele
 - Exprimarea (și motivarea) deciziilor de proiectare
 - Demonstrarea unor proprietăți generice ale limbajului
- Ca bază pentru demonstrarea corectitudinii programelor

Tipuri de semantică

- **Limbaj natural** – descriere textuală a efectelor

Tipuri de semantică

- **Limbaj natural** – descriere textuală a efectelor
- **Axiomatică** – descrierea folosind logică a efectelor unei instrucțiuni
 - $\vdash \{\varphi\} \text{cod} \{\psi\}$
 - modelează un program prin formulele logice pe care le satisface
 - utilă pentru demonstrarea corectitudinii

Tipuri de semantică

- **Limbaj natural** – descriere textuală a efectelor
- **Axiomatică** – descrierea folosind logică a efectelor unei instrucțiuni
 - $\vdash \{\varphi\} \text{cod} \{\psi\}$
 - modelează un program prin formulele logice pe care le satisface
 - utilă pentru demonstrarea corectitudinii
- **Denotațională** – asocierea unui obiect matematic (denotație)
 - $\llbracket \text{cod} \rrbracket$
 - modelează un program ca obiecte matematice
 - utilă pentru fundamente matematice

Tipuri de semantică

- **Limbaj natural** – descriere textuală a efectelor
- **Axiomatică** – descrierea folosind logică a efectelor unei instrucțiuni
 - $\vdash \{\varphi\} \text{cod} \{\psi\}$
 - modelează un program prin formulele logice pe care le satisface
 - utilă pentru demonstrarea corectitudinii
- **Denotațională** – asocierea unui obiect matematic (denotație)
 - $\llbracket \text{cod} \rrbracket$
 - modelează un program ca obiecte matematice
 - utilă pentru fundamente matematice
- **Operațională** – asocierea unei demonstrații pentru execuție
 - $\langle \text{cod}, \sigma \rangle \rightarrow \langle \text{cod}', \sigma' \rangle$
 - modelează un program prin execuția pe o mașină abstractă
 - utilă pentru implementarea de compilatoare și interpretoare

Tipuri de semantică

- **Limbaj natural** – descriere textuală a efectelor
- **Axiomatică** – descrierea folosind logică a efectelor unei instrucțiuni
 - $\vdash \{\varphi\} \text{cod} \{\psi\}$
 - modelează un program prin formulele logice pe care le satisface
 - utilă pentru demonstrarea corectitudinii
- **Denotațională** – asocierea unui obiect matematic (denotație)
 - $\llbracket \text{cod} \rrbracket$
 - modelează un program ca obiecte matematice
 - utilă pentru fundamente matematice
- **Operațională** – asocierea unei demonstrații pentru execuție
 - $\langle \text{cod}, \sigma \rangle \rightarrow \langle \text{cod}', \sigma' \rangle$
 - modelează un program prin execuția pe o mașină abstractă
 - utilă pentru implementarea de compilatoare și interpretoare
- **Statică** – asocierea unui sistem de tipuri care exclude programe eronate

Semantica small-step

Imagine de ansamblu

- **Semantica operatională** descrie cum se execută un program pe o mașină abstractă (ideală).

Imagine de ansamblu

- **Semantica operatională** descrie cum se execută un program pe o mașină abstractă (ideală).
- **Semantica operatională *small-step***
 - semantica structurală, a pașilor mici
 - descrie cum o execuție a programului avansează în funcție de reduceri succesive.

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

Imagine de ansamblu

- **Semantica operatională** descrie cum se execută un program pe o mașină abstractă (ideală).
- **Semantica operatională small-step**
 - semantica structurală, a pașilor mici
 - descrie cum o execuție a programului avansează în funcție de reduceri succesive.

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

- **Semantica operatională big-step**
 - semantică naturală, într-un pas mare

Starea execuției

- Starea execuției unui program IMP la un moment dat este dată de valorile deținute în acel moment de variabilele declarate în program.
- Formal, starea execuției unui program IMP la un moment dat este o funcție parțială (cu domeniu finit):

$$\sigma : Var \rightarrow Int$$

Starea execuției

- **Starea execuției** unui program IMP la un moment dat este dată de valorile deținute în acel moment de variabilele declarate în program.
- Formal, starea execuției unui program IMP la un moment dat este o **funcție parțială** (cu domeniu finit):

$$\sigma : Var \rightarrow Int$$

- **Notatii:**

- Descrierea funcției prin enumerare: $\sigma = n \mapsto 10, sum \mapsto 0$
- Funcția vidă \perp , nedefinită pentru nicio variabilă
- Obținerea valorii unei variabile: $\sigma(x)$
- Suprascrierea valorii unei variabile:

$$\sigma_{x \leftarrow v}(y) = \begin{cases} \sigma(y), & \text{dacă } y \neq x \\ v, & \text{dacă } y = x \end{cases}$$

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
 - Semantică Operațională Structurală
 - semantică prin tranziții
 - semantică prin reducere
- Defineste 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
- Defineste 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:
 $\langle x = 0 ; x = x + 1 ; , \perp \rangle \rightarrow \langle x = x + 1 ; , x \mapsto 0 \rangle$

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
 - Semantică Operațională Structurală
 - semantică prin tranziții
 - semantică prin reducere
- Defineste 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:

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

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
 - Semantică Operațională Structurală
 - semantică prin tranziții
 - semantică prin reducere
- Defineste 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:

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

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
 - Semantică Operațională Structurală
 - semantică prin tranziții
 - semantică prin reducere
- Defineste 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:

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

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
 - Semantică Operațională Structurală
 - semantică prin tranziții
 - semantică prin reducere
- Defineste 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:

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

- Cum definim această relație?

Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
 - Semantică Operațională Structurală
 - semantică prin tranziții
 - semantică prin reducere
- Defineste 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:

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

- Cum definim această relație? Prin inducție după elementele din sintaxă.

Redex. Reguli structurale. Axiome

□ Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

Redex. Reguli structurale. Axiome

□ Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

Redex. Reguli structurale. Axiome

□ Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

Redex. Reguli structurale. Axiome

□ Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

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

Redex. Reguli structurale. Axiome

□ Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

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

Axiome

- Realizează pasul computațional

Redex. Reguli structurale. Axiome

□ Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

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

Axiome

- Realizează pasul computațional

$\langle \text{if } (\text{true}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_1, \sigma \rangle$

Semantica expresiilor aritmetice

- Semantica unui întreg este o valoare
 - nu poate fi redex, deci nu avem regulă

- Semantica unei variabile

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

- Semantica adunării a două expresii aritmetice

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

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

Observatie: ordinea de evaluare a argumentelor este nespecificată.

Semantica expresiilor booleene

□ Semantica operatorului de comparatie

(LEQ-FALSE) $\langle i_1 = < i_2, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$ dacă $i_1 > i_2$

(LEQ-TRUE) $\langle i_1 = < i_2, \sigma \rangle \rightarrow \langle \text{true}, \sigma \rangle$ dacă $i_1 \leq i_2$

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

□ Semantica negatiei

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

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

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

Semantica expresiilor booleene

- Semantica operatorului de comparatie
- Semantica negatiei
- Semantica si-ului

(AND-FALSE) $\langle \text{and}(\text{false}, b_2), \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$

(AND-TRUE) $\langle \text{and}(\text{true}, b_2), \sigma \rangle \rightarrow \langle b_2, \sigma \rangle$

$$\frac{\langle b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle}{\langle \text{and}(b_1, b_2), \sigma \rangle \rightarrow \langle \text{and}(b'_1, b_2), \sigma \rangle}$$

Semantica compunerii si a blocurilor

□ Semantica blocurilor

(BLOCK) $\langle \{ s \} , \sigma \rangle \rightarrow \langle s , \sigma \rangle$

□ Semantica compunerii secventiale

(NEXT-STMT) $\langle \text{skip}; s_2 , \sigma \rangle \rightarrow \langle s_2 , \sigma \rangle$
 $\frac{\langle s_1 , \sigma \rangle \rightarrow \langle s'_1 , \sigma' \rangle}{\langle s_1 ; s_2 , \sigma \rangle \rightarrow \langle s'_1 ; s_2 , \sigma' \rangle}$

□ Semantica atribuirii

(ASGN) $\langle x = i , \sigma \rangle \rightarrow \langle \text{skip} , \sigma' \rangle$ dacă $\sigma' = \sigma_{x \leftarrow i}$

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

Semantica lui if

□ Semantica lui if

(IF-TRUE) $\langle \text{if}(\text{true}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_1, \sigma \rangle$

(IF-FALSE) $\langle \text{if}(\text{false}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_2, \sigma \rangle$

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

□ Semantica lui while

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

□ Semantica programelor

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

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

Semantica small-step a lui IMP

Execuție pas cu pas

$\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{P_{GM}}$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\begin{aligned} \langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle &\xrightarrow{\text{P}_{\text{GM}}} \\ \langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle &\xrightarrow{\text{W}_{\text{HILE}}} \end{aligned}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{P}_{\text{GM}}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{W}_{\text{HILE}}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{I}_0} \end{aligned}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{P}_{\text{GM}}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{W}_{\text{HILE}}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{I}_{\text{D}}} \\ &\langle \text{if } (0 \leq 3, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{LEQ-TRUE}} \end{aligned}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\begin{aligned} & \langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{PGM}} \\ & \langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{WHILE}} \\ & \langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{ID}} \\ & \langle \text{if } (0 \leq 3, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{LEQ-TRUE}} \\ & \langle \text{if } (\text{true}, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{IF-TRUE}} \end{aligned}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{P}_{\text{GM}}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{W}_{\text{HILE}}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{I}_{\text{D}}} \\ &\langle \text{if } (0 \leq 3, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{L}_{\text{EQ-TRUE}}} \\ &\langle \text{if } (\text{true}, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{I}_{\text{F-TRUE}}} \\ &\langle i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{I}_{\text{D}}} \\ &\dots \end{aligned}$$

O implementare a semanticii small-step

Semantica small-step

- Defineste cel mai mic pas de executie ca o relatie de tranzitie între configuratii:
 $\langle cod, \sigma \rangle \rightarrow \langle cod', \sigma' \rangle$ smallstep(Cod,S1,Cod',S2)
- Executia se obtine ca o succesiune de astfel de tranzitii.
- Starea executiei unui program IMP la un moment dat este o functie partială: $\sigma = n \mapsto 10, sum \mapsto 0$, 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

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

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

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

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):- ...
```

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 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 * si – se definesc similar.

Semantica expresiilor booleene

□ Semantica operatorului de comparatie

$\langle i_1 =< i_2, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$ *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):- ...
```

```
smallstepB(AE1 =< AE,S,AE2 =< AE,S):- ...
```

Semantica expresiilor Booleene

□ Semantica negatiei

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

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

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

Prolog

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

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

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

Semantica compunerii si a blocurilor

□ Semantica blocurilor

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

□ Semantica compunerii secventiale

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

Prolog

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

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

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

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) :- ...
```

Semantica lui if

□ Semantica lui if

$$\langle \text{if}(\text{true}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_1, \sigma \rangle$$
$$\langle \text{if}(\text{false}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_2, \sigma \rangle$$
$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle \text{if}(b, bl_1, bl_2), \sigma \rangle \rightarrow \langle \text{if}(b', bl_1, bl_2), \sigma \rangle}$$

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) :- ...
```

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) .
```

Executia 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
```


Executia programelor: trace

Putem defini o functie care ne permite să urmărim executia unui program în implementarea noastră?

Executia programelor: trace

Putem defini o functie care ne permite să urmărim executia 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,[]).
```

Executia programelor: trace_program

Exemplu

?- trace_program(pg2).

...

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

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)]

true .



Pe săptămâna viitoare!