

Topici speciale în logică și securitate I

Introducere în semantică

Ioana Leuştean

Master anul II, Sem. I, 2019-2020

Cuprins

- ① Semantica programelor - idei generale
- ② Semantica axiomatică
- ③ Semantica denotațională
- ④ Semantica operațională (small-step)
- ⑤ Semantica big-step

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?
 - De cele mai multe ori se dă din umeri și se spune că Practica e suficientă

Acest material are la bază cursul introductiv:

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

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
 - Explicarea (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
 - Explicarea (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

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

Tipuri de semantică

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

Limbajul IMP

IMP este un limbaj IMPerativ foarte simplu.

Ce conține:

- Expresii

- Aritmetice
- Booleene

$x + 3$
 $(x > 7)$

- Blocuri de instrucțiuni

- De atribuire
- Condiționale
- De ciclare

$x = 5;$
`if (x > 7) {x =5; } else {x = 0;}`
`while (x > 7) {x = x - 1;}`

Ce nu conține:

- Expresii cu efecte laterale
- Proceduri și funcții
- Schimbări abrupte de control

Limbajul IMP

Exemplu

Un program în limbajul IMP

```
int x = 10;  
int y = 1;  
while (0 < x) {  
    y = y * x;  
    x = x + -1;  
}
```

Sintaxa BNF a limbajului IMP

$$E ::= n \mid x \\ \mid E + E \mid E * E$$
$$B ::= \text{true} \mid \text{false} \\ \mid E \leq E \mid E < E \\ \mid ! B \mid B \&\& B$$
$$C ::= \{ C \} \mid \{ \}$$
$$C ::= C \mid C \ C \\ \mid x = E ; \\ \mid \text{if } (B) \ C \text{ else } C \\ \mid \text{while } (B) \ C$$
$$P ::= \text{int } x = n ; P \mid C$$

Atribuirea: $x = \text{expr}$

- Expresia este evaluată în starea curentă a programului
- Variabilei i se atribuie valoarea calculată, înlocuind valoarea precedentă a acelei variabile.

Atribuirea: $x = \text{expr}$

- Expresia este evaluată în starea curentă a programului
- Variabilei i se atribuie valoarea calculată, înlocuind valoarea precedentă a acelei variabile.

Avantaje și dezavantaje

- + Ușor de prezentat
- Potențial ambiguă
- Imposibil de procesat automat

Semantica axiomatică

Semantica Axiomatică

Logica Floyd-Hoare

- Inventată de 1969 Tony Hoare în 1969 (inspirată de rezultatele lui Robert Floyd).
- Definește triplete (**triplete Hoare**) de forma

$$\{Pre\} S \{Post\}$$

unde:

- S este o instrucțiune (Stmt)
- Pre (precondiție), respectiv $Post$ (postcondiție) sunt aserțiuni logice asupra stării sistemului înaintea, respectiv după execuția lui S
- Limbajul aserțiunilor este un limbaj de ordinul I.
- Tripletul $\{Pre\} S \{Post\}$ este (parțial) *corect* dacă:
 - dacă programul se execută dintr-o stare inițială care satisface Pre
 - și execuția se termină
 - atunci se ajunge într-o stare finală care satisface $Post$.

Semantica Axiomatică

Logica Floyd-Hoare

Definește triplete (**triplete Hoare**) de forma

$$\{Pre\} S \{Post\}$$

- Triplețul $\{Pre\} S \{Post\}$ este (parțial) *corect* dacă:
 - dacă programul se execută dintr-o stare inițială care satisface *Pre*
 - și execuția se termină
 - atunci se ajunge într-o stare finală care satisface *Post*.

Exemplu

- $\{x = 1\} x = x+1 \{x = 2\}$ este corect
- $\{x = 1\} x = x+1 \{x = 3\}$ **nu** este corect
- $\{\top\} \text{if } (x \leq y) \ z=x; \text{ else } z=y; \{z = \min(x, y)\}$ este corect

Logica Floyd-Hoare a fost studiată la cursul "Verificarea programelor".

Semantica denotațională

Semantica denotațională

- Introdusă de Christopher Strachey și Dana Scott (1970)
- Semantica operațională, ca un interpretor, descrie **cum** să evaluăm un program.
- **Semantica denotațională**, ca un compilator, descrie o traducere a limbajului într-un limbaj diferit cu semantică cunoscută, anume matematica.
- Semantica denotațională definește ce înseamnă un program ca o funcție matematică.

Semantica denotațională

- Definim stările memoriei ca fiind funcții parțiale de la mulțimea identificatorilor la mulțimea valorilor:

$$State = Id \rightarrow \mathbb{Z}$$

- Asociem fiecărei categorii sintactice o categorie semantică.
- Fiecare construcție sintactică va avea o denotație (interpretare) în categoria semantică respectivă.

Semantica denotațională

- Definim stările memoriei ca fiind funcții parțiale de la mulțimea identificatorilor la mulțimea valorilor:

$$State = Id \rightarrow \mathbb{Z}$$

- Asociem fiecărei categorii sintactice o categorie semantică.
- Fiecare construcție sintactică va avea o denotație (interpretare) în categoria semantică respectivă. De exemplu:
 - denotația unei expresii aritmetice este o funcție parțială de la mulțimea stărilor memoriei la mulțimea valorilor (\mathbb{Z}):

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

- denotația unei instrucțiuni este o funcție parțială de la mulțimea stărilor memoriei la mulțimea stărilor memoriei:

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

$State = Id \rightarrow \mathbb{Z}$

$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$

$[[_]] : Stmt \rightarrow (State \rightarrow State)$

Atribuirea: $x = \text{expr}$

- Asociem expresiilor aritmetice funcții de la starea memoriei la valori:
- Asociem instrucțiunilor funcții de la starea memoriei la starea (următoare) a memoriei.

$$State = Id \rightarrow \mathbb{Z}$$

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

Atribuirea: $x = \text{expr}$

- Asociem expresiilor aritmetice funcții de la starea memoriei la valori:
 - Funcția constantă $[[1]](s) = 1$
 - Funcția care selectează valoarea unui identificator $[[x]](s) = s(x)$
 - „Morfismul de adunare” $[[e1 + e2]](s) = [[e1]](s) + [[e2]](s)$.
- Asociem instrucțiunilor funcții de la starea memoriei la starea (următoare) a memoriei.
 - $[[x = e]](s)(y) = \begin{cases} s(y), & \text{dacă } y \neq x \\ [[e]](s), & \text{dacă } y = x \end{cases}$

Semantica denotațională a limbajului IMP

$$State = Id \rightarrow \mathbb{Z}$$

- Domenii semantice:

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : BExp \rightarrow (State \rightarrow \{T, F\})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

Semantica denotațională a limbajului IMP

$$State = Id \rightarrow \mathbb{Z}$$

- Domenii semantice:

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : BExp \rightarrow (State \rightarrow \{T, F\})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

- Semantica denotațională este compozițională:

- semantica expresiilor aritmetice

$$[[n]](s) = n$$

$$[[x]](s) = s(x)$$

$$[[e1 + e2]](s) = [[e1]](s) + [[e2]](s)$$

Semantica denotațională a limbajului IMP

$$State = Id \rightarrow \mathbb{Z}$$

- Domenii semantice:

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : BExp \rightarrow (State \rightarrow \{T, F\})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

- Semantica denotațională este compozițională:

- semantica expresiilor aritmetice

$$[[n]](s) = n$$

$$[[x]](s) = s(x)$$

$$[[e1 + e2]](s) = [[e1]](s) + [[e2]](s)$$

- semantica expresiilor booleene

$$[[true]](s) = T, [[false]](s) = F$$

$$[[!b]](s) = \neg b$$

$$[[e1 <= e2]](s) = [[e1]](s) <= [[e2]](s)$$

Semantica denotațională a limbajului IMP

$$State = Id \rightarrow \mathbb{Z}$$

- Domenii semantice:

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : BExp \rightarrow (State \rightarrow \{T, F\})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

Semantica denotațională a limbajului IMP

$$State = Id \rightarrow \mathbb{Z}$$

- Domenii semantice:

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : BExp \rightarrow (State \rightarrow \{T, F\})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

- Semantica instrucțiunilor:

$$[[skip]] = id$$

$$[[c1; c2]] = [[c2]] \circ [[c1]]$$

$$[[x = e]](s)(y) = \begin{cases} s(y), & \text{dacă } y \neq x \\ [[e]](s), & \text{dacă } y = x \end{cases}$$

Semantica denotațională a limbajului IMP

$$State = Id \rightarrow \mathbb{Z}$$

- Domenii semantice:

$$[[_]] : AExp \rightarrow (State \rightarrow \mathbb{Z})$$

$$[[_]] : BExp \rightarrow (State \rightarrow \{T, F\})$$

$$[[_]] : Stmt \rightarrow (State \rightarrow State)$$

- Semantica instrucțiunilor:

$$[[skip]] = id$$

$$[[c1; c2]] = [[c2]] \circ [[c1]]$$

$$[[x = e]](s)(y) = \begin{cases} s(y), & \text{dacă } y \neq x \\ [[e]](s), & \text{dacă } y = x \end{cases}$$

$$[[if (b) c1 else c2]](s) = \begin{cases} [[c1]](s), & \text{dacă } [[b]](s) = T \\ [[c2]](s), & \text{dacă } [[b]](s) = F \end{cases}$$

Exemplu

if ($x \leq y$) $z=x$; else $z=y$;

$$[[pgm]](s) = \begin{cases} [[z = x;]](s), & \text{dacă } [[x \leq y]](s) = T \\ [[z = y;]](s), & \text{dacă } [[x \leq y]](s) = F \end{cases}$$

Exemplu

if ($x \leq y$) $z=x$; else $z=y$;

$$[[pgm]](s) = \begin{cases} [[z = x;]](s), & \text{dacă } [[x \leq y]](s) = T \\ [[z = y;]](s), & \text{dacă } [[x \leq y]](s) = F \end{cases}$$

$$[[pgm]](s)(v) = \begin{cases} s(v), & \text{dacă } s(x) \leq s(y), v \neq z \\ s(x), & \text{dacă } s(x) \leq s(y), v = z \\ s(v), & \text{dacă } s(x) > s(y), v \neq z \\ s(y), & \text{dacă } s(x) > s(y), v = z \end{cases}$$

Semantica denotațională a limbajului IMP

Exemplu

if ($x \leq y$) $z=x$; else $z=y$;

$$[[pgm]](s) = \begin{cases} [[z = x;]](s), & \text{dacă } [[x \leq y]](s) = T \\ [[z = y;]](s), & \text{dacă } [[x \leq y]](s) = F \end{cases}$$

$$[[pgm]](s)(v) = \begin{cases} s(v), & \text{dacă } s(x) \leq s(y), v \neq z \\ s(x), & \text{dacă } s(x) \leq s(y), v = z \\ s(v), & \text{dacă } s(x) > s(y), v \neq z \\ s(y), & \text{dacă } s(x) > s(y), v = z \end{cases}$$

Cum definim semantica denotațională pentru `while`?

Mulțimea funcțiilor parțiale

Fie X și Y două mulțimi.

- $Pfn(X, Y)$ mulțimea funcțiilor parțiale de la X la Y , adică $Pfn(X, Y) = X \rightharpoonup Y$
- Pentru $f \in Pfn(X, Y)$ notăm cu $dom(f)$ mulțimea elementelor din X pentru care funcția este definită.

Atunci $dom(f) \subseteq X$ și $f|_{dom(f)} : dom(f) \rightarrow Y$ este funcție.

Mulțimea funcțiilor parțiale

Fie X și Y două mulțimi.

- $Pfn(X, Y)$ mulțimea funcțiilor parțiale de la X la Y , adică $Pfn(X, Y) = X \multimap Y$
- Pentru $f \in Pfn(X, Y)$ notăm cu $dom(f)$ mulțimea elementelor din X pentru care funcția este definită.

Atunci $dom(f) \subseteq X$ și $f|_{dom(f)} : dom(f) \rightarrow Y$ este funcție.

- Fie $\perp : X \multimap Y$ unica funcție cu $dom(\perp) = \emptyset$ (funcția care nu este definită în nici un punct).
- Definim pe $Pfn(X, Y)$ următoarea relație:

$$f \sqsubseteq g \text{ dacă și numai dacă } dom(f) \subseteq dom(g) \text{ și } g|_{dom(f)} = f|_{dom(f)}$$

Mulțimea funcțiilor parțiale

Fie X și Y două mulțimi.

- $Pfn(X, Y)$ mulțimea funcțiilor parțiale de la X la Y , adică $Pfn(X, Y) = X \multimap Y$
- Pentru $f \in Pfn(X, Y)$ notăm cu $dom(f)$ mulțimea elementelor din X pentru care funcția este definită.

Atunci $dom(f) \subseteq X$ și $f|_{dom(f)} : dom(f) \rightarrow Y$ este funcție.

- Fie $\perp : X \multimap Y$ unica funcție cu $dom(\perp) = \emptyset$ (funcția care nu este definită în nici un punct).
- Definim pe $Pfn(X, Y)$ următoarea relație:

$$f \sqsubseteq g \text{ dacă și numai dacă } dom(f) \subseteq dom(g) \text{ și } g|_{dom(f)} = f|_{dom(f)}$$

$(Pfn(X, Y), \sqsubseteq, \perp)$ este CPO

(mulțime parțial ordonată completă în care \perp este cel mai mic element)

$(Pfn(X, Y), \sqsubseteq, \perp)$ este CPO

Exemplu

Definim $\mathbf{F} : Pfn(\mathbb{N}, \mathbb{N}) \rightarrow Pfn(\mathbb{N}, \mathbb{N})$ prin

$$\mathbf{F}(g)(k) = \begin{cases} 1, & \text{dacă } k = 0, \\ k * g(k-1) & \text{dacă } k > 0 \text{ și } (k-1) \in \text{dom}(g), \\ \text{nedefinit}, & \text{altfel} \end{cases}$$

- \mathbf{F} este o funcție continuă,

$(Pfn(X, Y), \sqsubseteq, \perp)$ este CPO

Exemplu

Definim $\mathbf{F} : Pfn(\mathbb{N}, \mathbb{N}) \rightarrow Pfn(\mathbb{N}, \mathbb{N})$ prin

$$\mathbf{F}(g)(k) = \begin{cases} 1, & \text{dacă } k = 0, \\ k * g(k-1) & \text{dacă } k > 0 \text{ și } (k-1) \in \text{dom}(g), \\ \text{nedefinit}, & \text{altfel} \end{cases}$$

- \mathbf{F} este o funcție continuă, deci putem aplica

- Teorema Knaster-Tarski

Fie $g_n = \mathbf{F}^n(\perp)$ și $f = \bigvee_n g_n$.

Știm că f este cel mai mic punct fix al funcției \mathbf{F} , deci $\mathbf{F}(f) = f$.

$(Pfn(X, Y), \sqsubseteq, \perp)$ este CPO

Exemplu

Definim $\mathbf{F} : Pfn(\mathbb{N}, \mathbb{N}) \rightarrow Pfn(\mathbb{N}, \mathbb{N})$ prin

$$\mathbf{F}(g)(k) = \begin{cases} 1, & \text{dacă } k = 0, \\ k * g(k-1) & \text{dacă } k > 0 \text{ și } (k-1) \in \text{dom}(g), \\ \text{nedefinit}, & \text{altfel} \end{cases}$$

- \mathbf{F} este o funcție continuă, deci putem aplica
- Teorema Knaster-Tarski
Fie $g_n = \mathbf{F}^n(\perp)$ și $f = \bigvee_n g_n$.
Știm că f este cel mai mic punct fix al funcției \mathbf{F} , deci $\mathbf{F}(f) = f$.
- Demonstrăm prin inducție după n că:
 $\text{dom}(g_n) = \{0, \dots, n\}$ și $g_n(k) = k!$ oricare $k \in \text{dom}(g_n)$
- $f: \mathbb{N} \rightarrow \mathbb{N}$ este funcția factorial.

Semantica denotațională pentru while

while (b) c

- Definim $\mathbf{F} : Pfn(State, State) \rightarrow Pfn(State, State)$ prin
- \mathbf{F} este continuă
- Teorema Knaster-Tarski: $fix(\mathbf{F}) = \bigcup_n \mathbf{F}^n(\perp)$

Semantica denotațională pentru while

while (b) c

- Definim $\mathbf{F} : Pfn(State, State) \rightarrow Pfn(State, State)$ prin

$$\mathbf{F}(g)(s) = \begin{cases} g([[c]](s)) & \text{dacă } [[b]](s) = T \\ s & \text{dacă } [[b]](s) = F \\ \text{nedefinit,} & \text{altfel} \end{cases}$$

- \mathbf{F} este continuă
- Teorema Knaster-Tarski: $fix(\mathbf{F}) = \bigcup_n \mathbf{F}^n(\perp)$

Semantica denotațională pentru while

while (b) c

- Definim $\mathbf{F} : Pfn(State, State) \rightarrow Pfn(State, State)$ prin

$$\mathbf{F}(g)(s) = \begin{cases} g([[c]](s)) & \text{dacă } [[b]](s) = T \\ s & \text{dacă } [[b]](s) = F \\ \text{nedefinit,} & \text{altfel} \end{cases}$$

- \mathbf{F} este continuă
- Teorema Knaster-Tarski: $fix(\mathbf{F}) = \bigcup_n \mathbf{F}^n(\perp)$
- Semantica denotațională:

$[[_]] : Stmt \rightarrow (State \rightarrow State)$

$[[\text{while } (b) \ c]](s) = fix(\mathbf{F})(s)$

Avantaje și dezavantaje

- + Formală, matematică, foarte precisă
- + Compozițională (morfisme și compuneri de funcții)
- Domeniile devin din ce în ce mai complexe.

Semantica operațională (small-step)

Imagine de ansamblu

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

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

$$\sigma : Var \rightarrow Int$$

- **Notății:**
 - 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
- 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:

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

- Execuția se obține ca o succesiune de astfel de tranziții:
 $\langle \text{int } x = 0 ; x = x + 1 ; , \perp \rangle \rightarrow \langle x = x + 1 ; , x \mapsto 0 \rangle$

Semantica small-step

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

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

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:

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

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

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

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:

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

- 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

- 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:

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

- Cum definim această relație?

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:

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

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

- Expresie reductibilă (redex)
 - Fragmentul de sintaxă care va fi procesat la pasul următor
`if (0 <= 5 + 7 * x) { r = 1 ; } else { 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 ; } else { 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 ; } else { 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) { $r = 1$; } else { $r = 0$; }

- Reguli structurale

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

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

Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

if (0 <= 5 + 7 * x) { r = 1 ; } else { r = 0 ; }

- Reguli structurale

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

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

- 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 ; } else { r = 0 ; }

- Reguli structurale

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

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

- Axiome

- Realizează pasul computațional

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

Sintaxa BNF a limbajului IMP

$$E ::= n \mid x \\ \mid E + E \mid E * E$$
$$B ::= \text{true} \mid \text{false} \\ \mid E \leq E \\ \mid ! B \mid B \&\& B$$
$$C ::= \{ C \} \mid \{ \}$$
$$C ::= C \mid C \ C \\ \mid x = E ; \\ \mid \text{if} (B) C \text{ else } C \\ \mid \text{while} (B) C$$
$$P ::= \text{int } x = n ; P \mid C$$

Semantica small-step a lui IMP

Semantica expresiilor aritmetice

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

Semantica expresiilor aritmetice

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

$$(Id) \quad \langle x, \sigma \rangle \rightarrow \langle i, \sigma \rangle \quad \text{dacă } i = \sigma(x)$$

Semantica expresiilor aritmetice

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

- Semantica unei variabile

$$(\text{ID}) \quad \langle x, \sigma \rangle \rightarrow \langle i, \sigma \rangle \quad \text{dacă } i = \sigma(x)$$

- Semantica adunării a două expresii aritmetice

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

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

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

Semantica expresiilor aritmetice

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

- Semantica unei variabile

$$(\text{ID}) \quad \langle x, \sigma \rangle \rightarrow \langle i, \sigma \rangle \quad \text{dacă } i = \sigma(x)$$

- Semantica adunării a două expresii aritmetice

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

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

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

- Semantica înmulțirii a două expresii aritmetice – similar

Semantica small-step a lui IMP

Semantica expresiilor Booleene

- Semantica constantelor Booleene sunt valori
 - nu pot fi redex, deci nu avem reguli

Semantica small-step a lui IMP

Semantica expresiilor Booleene

- Semantica constantelor Booleene sunt valori
 - nu pot fi redex, deci nu avem reguli
- Semantica operatorului de comparație

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

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

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

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

Semantica small-step a lui IMP

- Semantica negației

Semantica small-step a lui IMP

- Semantica negației

(!-TRUE) $\langle !\text{true}, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$

(!-FALSE) $\langle !\text{false}, \sigma \rangle \rightarrow \langle \text{true}, \sigma \rangle$

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

Semantica small-step a lui IMP

- Semantica negației

(!-TRUE) $\langle \text{!true}, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$

(!-FALSE) $\langle \text{!false}, \sigma \rangle \rightarrow \langle \text{true}, \sigma \rangle$

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle \text{! } a, \sigma \rangle \rightarrow \langle \text{! } a', \sigma \rangle}$$

- Semantica și-ului

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

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

$$\frac{\langle b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle}{\langle b_1 \&\& b_2, \sigma \rangle \rightarrow \langle b'_1 \&\& b_2, \sigma \rangle}$$

Semantica small-step a lui IMP

Semantica comenzilor

- Semantica blocurilor

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

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

Atenție! O instrucțiune poate modifica starea curentă!

Semantica small-step a lui IMP

Semantica comenzilor

- Semantica blocurilor

$$(\text{BLOCK-END}) \quad \langle \{\{\}\} , \sigma \rangle \rightarrow \langle \{\} , \sigma \rangle$$

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

Atenție! O instrucțiune poate modifica starea curentă!

- Semantica compunerii secvențiale

$$(\text{NEXT-STMT}) \quad \langle \{\} s_2 , \sigma \rangle \rightarrow \langle s_2 , \sigma \rangle$$

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

Semantica small-step a lui IMP

- Semantica atribuirii

(ASGN) $\langle x = i ; , \sigma \rangle \rightarrow \langle \{\} , \sigma' \rangle$ 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 small-step a lui IMP

- Semantica atribuirii

(ASGN) $\langle x = i ; , \sigma \rangle \rightarrow \langle \{\} , \sigma' \rangle$ 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

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

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

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

Semantica small-step a lui IMP

- Semantica lui while

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

Semantica small-step a lui IMP

- Semantica lui while

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

- Semantica inițializărilor

(INIT) $\langle \text{int } x = i ; p, \sigma \rangle \rightarrow \langle p, \sigma' \rangle \quad \text{dacă } \sigma' = \sigma_{x \leftarrow i}$

Semantica small-step a lui IMP

Execuție pas cu pas

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

$\xrightarrow{\text{INIT}}$

Semantica small-step a lui IMP

Execuție pas cu pas

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

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

$$\begin{array}{c} \xrightarrow{\text{INIT}} \\ \xrightarrow{\text{WHILE}} \end{array}$$

Semantica small-step a lui IMP

Execuție pas cu pas

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

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

$\langle \text{if } (0 \leq i) \{ \{ i = i + -4 ; \} \quad \text{while } (0 \leq i) \{ i = i + -4 ; \} \quad \text{else } \{ \} \} , i \mapsto 3 \rangle$

$\xrightarrow{\text{INIT}}$
 $\xrightarrow{\text{WHILE}}$
 $\xrightarrow{\text{ID}}$

Semantica small-step a lui IMP

Execuție pas cu pas

$\langle \text{int } i = 3 ; \text{while } (0 \leq i) \{ i = i + -4 ; \} , \perp \rangle$	$\xrightarrow{\text{INIT}}$
$\langle \text{while } (0 \leq i) \{ i = i + -4 ; \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{WHILE}}$
$\langle \text{if } (0 \leq i) \{ \{ i = i + -4 ; \} \quad , i \mapsto 3 \}$ $\text{while } (0 \leq i) \{ i = i + -4 ; \}$ $\} \text{ else } \{ \}$	$\xrightarrow{\text{ID}}$
$\langle \text{if } (0 \leq 3) \{ \{ i = i + -4 ; \} \quad , i \mapsto 3 \}$ $\text{while } (0 \leq i) \{ i = i + -4 ; \}$ $\} \text{ else } \{ \}$	$\xrightarrow{\text{LEQ-TRUE}}$

Semantica small-step a lui IMP

Execuție pas cu pas

$\langle \text{int } i = 3 ; \text{while } (0 \leq i) \{ i = i + -4 ; \} , \perp \rangle$	$\xrightarrow{\text{INIT}}$
$\langle \text{while } (0 \leq i) \{ i = i + -4 ; \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{WHILE}}$
$\langle \text{if } (0 \leq i) \{ \{ i = i + -4 ; \} \quad \text{while } (0 \leq i) \{ i = i + -4 ; \} \quad \text{else } \{ \} \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{ID}}$
$\langle \text{if } (0 \leq 3) \{ \{ i = i + -4 ; \} \quad \text{while } (0 \leq i) \{ i = i + -4 ; \} \quad \text{else } \{ \} \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{LEQ-TRUE}}$
$\langle \text{if } (\text{true}) \{ \{ i = i + -4 ; \} \quad \text{while } (0 \leq i) \{ i = i + -4 ; \} \quad \text{else } \{ \} \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{IF-TRUE}}$

Semantica small-step a lui IMP

Execuție pas cu pas

$\langle \text{int } i = 3 ; \text{while } (0 \leq i) \{ i = i + -4 ; \} , \perp \rangle$	$\xrightarrow{\text{INIT}}$
$\langle \text{while } (0 \leq i) \{ i = i + -4 ; \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{WHILE}}$
$\langle \text{if } (0 \leq i) \{ \{ i = i + -4 ; \}$ $\text{while } (0 \leq i) \{ i = i + -4 ; \}$ $\} \text{ else } \{ \}$ $\} , i \mapsto 3 \rangle$	$\xrightarrow{\text{ID}}$
$\langle \text{if } (0 \leq 3) \{ \{ i = i + -4 ; \}$ $\text{while } (0 \leq i) \{ i = i + -4 ; \}$ $\} \text{ else } \{ \}$ $\} , i \mapsto 3 \rangle$	$\xrightarrow{\text{LEQ-TRUE}}$
$\langle \text{if } (\text{true}) \{ \{ i = i + -4 ; \}$ $\text{while } (0 \leq i) \{ i = i + -4 ; \}$ $\} \text{ else } \{ \}$ $\} , i \mapsto 3 \rangle$	$\xrightarrow{\text{IF-TRUE}}$
$\langle \{ \{ i = i + -4 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{ID}}$

Semantica small-step a lui IMP

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

Semantica small-step a lui IMP

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

$\xrightarrow{\text{ADD}}$

$\langle \{ \{ i = -1 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$

$\xrightarrow{\text{ASGN}}$

Semantica small-step a lui IMP

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

$\langle \{ \{ i = -1 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$

$\langle \{ \{ \{ \} \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

ADD

→

ASGN

→

BLOCK-END

→

Semantica small-step a lui IMP

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

$\langle \{ \{ i = -1 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$

$\langle \{ \{ \{ \} \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

$\langle \{ \{ \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

ADD

\longrightarrow

ASGN

\longrightarrow

BLOCK-END

\longrightarrow

NEXT-STMT

\longrightarrow

Semantica small-step a lui IMP

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

$\langle \{ \{ i = -1 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$

$\langle \{ \{ \{ \} \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

$\langle \{ \{ \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

$\langle \{ \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

ADD

→

ASGN

→

BLOCK-END

→

NEXT-STMT

→

WHILE

→

Semantica small-step a lui IMP

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

$\langle \{ \{ i = -1 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$

$\langle \{ \{ \{ \} \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

$\langle \{ \{ \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

$\langle \{ \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$

$\langle \{ \text{if } (0 \leq i) \{ \{ i = i + -4 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} \text{ else } \{ \} \} , i \mapsto -1 \rangle$

ADD

ASGN

BLOCK-END

NEXT-STMT

WHILE

ID

Semantica small-step a lui IMP

$\langle \{ \{ i = 3 + -4 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{ADD}}$
$\langle \{ \{ i = -1 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto 3 \rangle$	$\xrightarrow{\text{ASGN}}$
$\langle \{ \{ \{ \} \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$	$\xrightarrow{\text{BLOCK-END}}$
$\langle \{ \{ \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$	$\xrightarrow{\text{NEXT-STMT}}$
$\langle \{ \text{while } (0 \leq i) \{ i = i + -4 ; \} \} , i \mapsto -1 \rangle$	$\xrightarrow{\text{WHILE}}$
$\langle \{ \text{if } (0 \leq i) \{ \{ i = i + -4 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} \text{ else } \{ \} \} , i \mapsto -1 \rangle$	$\xrightarrow{\text{ID}}$
$\langle \{ \text{if } (0 \leq -1) \{ \{ i = i + -4 ; \} \text{while } (0 \leq i) \{ i = i + -4 ; \} \} \text{ else } \{ \} \} , i \mapsto -1 \rangle$	$\xrightarrow{\text{LEQ-FALSE}}$

Semantica small-step a lui IMP

$\langle \{ \text{if (false) } \{ \{ i = i + -4 ; \}$
 $\text{while } (0 \leq i) \{ i = i + -4 ; \}$
 $\} \text{ else } \{ \} \} , i \mapsto -1 \rangle \xrightarrow{\text{IF-FALSE}}$

Semantica small-step a lui IMP

$$\begin{array}{lcl} \langle \{ \text{if (false) } \{ \{ i = i + -4 ; \} & , i \mapsto -1 \rangle & \xrightarrow{\text{IF-FALSE}} \\ \quad \text{while } (0 \leq i) \{ i = i + -4 ; \} & & \\ \quad \} \text{ else } \{ \} \} & & \\ \langle \{ \{ \} \} , i \mapsto -1 \rangle & & \xrightarrow{\text{BLOCK-END}} \end{array}$$

Semantica small-step a lui IMP

$$\begin{array}{lcl} \langle \{ \text{if (false) } \{ \{ i = i + -4 ; \} & , i \mapsto -1 \rangle & \xrightarrow{\text{IF-FALSE}} \\ \quad \text{while } (0 \leq i) \{ i = i + -4 ; \} & & \\ \quad \} \text{ else } \{ \} \} & & \\ \langle \{ \{ \} \} , i \mapsto -1 \rangle & & \xrightarrow{\text{BLOCK-END}} \\ \langle \{ \} , i \mapsto -1 \rangle & & \end{array}$$

Semantica big-step

Semantica big-step

- Introdusă de Gilles Kahn (1987)
- Denumiri alternative:
 - semantică relațională
 - semantică naturală
- Relaționează fragmente de program într-o stare cu valoarea corespunzătoare evaluării lor în acea stare
 - Expresiile aritmetice se evaluează la întregi: $\langle a, \sigma \rangle \Downarrow \langle i \rangle$
 - Expresiile Booleene se evaluează la *true/false*: $\langle b, \sigma \rangle \Downarrow \langle t \rangle$
 - Instrucțiunile se evaluează la stări: $\langle ins, \sigma \rangle \Downarrow \langle \sigma' \rangle$
 - Blocurile se evaluează la stări: $\langle bl, \sigma \rangle \Downarrow \langle \sigma' \rangle$
 - Programul se evaluează la o stare: $\langle p \rangle \Downarrow \langle \sigma \rangle$ sau $\langle p, \sigma \rangle \Downarrow \langle \sigma' \rangle$
- Reguli structurale, având ca premise secvenți corespunzători subtermenilor.
- Valoarea este obținută într-un **singur pas**.

Exemplu

- $\langle 3 + x, (x \mapsto 5, y \mapsto 7) \rangle \Downarrow \langle 8 \rangle$
- $\langle x = 3 + y, (x \mapsto 5, y \mapsto 7) \rangle \Downarrow \langle x \mapsto 10, y \mapsto 7 \rangle$

Expresii aritmetice

$$(INT) \quad \langle i, \sigma \rangle \Downarrow \langle \hat{i} \rangle$$

$$(ID) \quad \langle x, \sigma \rangle \Downarrow \langle \hat{i} \rangle \quad \text{dacă } i = \sigma(x)$$

$$(ADD) \quad \frac{\langle a_1, \sigma \rangle \Downarrow \langle \hat{i}_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle \hat{i}_2 \rangle}{\langle a_1 + a_2, \sigma \rangle \Downarrow \langle \hat{i} \rangle} \quad \text{dacă } i = \hat{i}_1 + \hat{i}_2$$

$$(MUL) \quad \frac{\langle a_1, \sigma \rangle \Downarrow \langle \hat{i}_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle \hat{i}_2 \rangle}{\langle a_1 * a_2, \sigma \rangle \Downarrow \langle \hat{i} \rangle} \quad \text{dacă } i = \hat{i}_1 * \hat{i}_2$$

Semantica big-step a lui IMP

Expresii booleene

$$(BOL) \quad \langle t, \sigma \rangle \Downarrow \langle t \rangle$$

$$(CMP) \quad \frac{\langle a_1, \sigma \rangle \Downarrow \langle i_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle i_2 \rangle}{\langle a_1 \leq a_2, \sigma \rangle \Downarrow \langle t \rangle} \quad \text{dacă } t = i_1 \leq i_2$$

$$(NOT-TRUE) \quad \frac{\langle b, \sigma \rangle \Downarrow \langle false \rangle}{\langle ! b, \sigma \rangle \Downarrow \langle true \rangle}$$

$$(NOT-FALSE) \quad \frac{\langle b, \sigma \rangle \Downarrow \langle true \rangle}{\langle ! b, \sigma \rangle \Downarrow \langle false \rangle}$$

$$(AND-TRUE) \quad \frac{\langle b_1, \sigma \rangle \Downarrow \langle true \rangle \quad \langle b_2, \sigma \rangle \Downarrow \langle t \rangle}{\langle b_1 \&\& b_2, \sigma \rangle \Downarrow \langle t \rangle}$$

$$(AND-FALSE) \quad \frac{\langle b_1, \sigma \rangle \Downarrow \langle false \rangle}{\langle b_1 \&\& b_2, \sigma \rangle \Downarrow \langle false \rangle}$$

Semantica big-step a lui IMP

Instrucțiuni simple

$$(\text{SECV}) \quad \frac{\langle s_1, \sigma \rangle \Downarrow \langle \sigma' \rangle \quad \langle s_2, \sigma' \rangle \Downarrow \langle \sigma'' \rangle}{\langle s_1 \ s_2, \sigma \rangle \Downarrow \langle \sigma'' \rangle}$$

$$(\text{ASGN}) \quad \frac{\langle a, \sigma \rangle \Downarrow \langle i \rangle}{\langle x = a ; , \sigma \rangle \Downarrow \langle \sigma' \rangle} \quad \text{dacă } \sigma' = \sigma[i/x]$$

$$(\text{IF-TRUE}) \quad \frac{\langle b, \sigma \rangle \Downarrow \langle \text{true} \rangle \quad \langle bl_1, \sigma \rangle \Downarrow \langle \sigma_1 \rangle}{\langle \text{if } (b) \ bl_1 \text{ else } bl_2, \sigma \rangle \Downarrow \langle \sigma_1 \rangle}$$

$$(\text{IF-FALSE}) \quad \frac{\langle b, \sigma \rangle \Downarrow \langle \text{false} \rangle \quad \langle bl_2, \sigma \rangle \Downarrow \langle \sigma_2 \rangle}{\langle \text{if } (b) \ bl_1 \text{ else } bl_2, \sigma \rangle \Downarrow \langle \sigma_2 \rangle}$$

Semantica big-step a lui IMP

Blocuri și instrucțiuni de ciclare

$$(\text{SKIP}) \quad \langle \{\} , \sigma \rangle \Downarrow \langle \sigma \rangle$$

$$(\text{BLOCK}) \quad \frac{\langle s , \sigma \rangle \Downarrow \langle \sigma' \rangle}{\langle \{ s \} , \sigma \rangle \Downarrow \langle \sigma' \rangle}$$

$$(\text{WHILE-TRUE}) \quad \frac{\langle b , \sigma \rangle \Downarrow \langle \text{true} \rangle \quad \langle bl , \sigma \rangle \Downarrow \langle \sigma' \rangle \quad \langle \text{while } (b) \text{ } bl , \sigma' \rangle \Downarrow \langle \sigma'' \rangle}{\langle \text{while } (b) \text{ } bl , \sigma \rangle \Downarrow \langle \sigma'' \rangle}$$

$$(\text{WHILE-FALSE}) \quad \frac{\langle b , \sigma \rangle \Downarrow \langle \text{false} \rangle}{\langle \text{while } (b) \text{ } bl , \sigma \rangle \Downarrow \langle \sigma \rangle}$$

Semantica big-step a lui IMP

Inițializări. Semantica programului

$$(\text{INIT}) \quad \frac{\langle p, \sigma' \rangle \Downarrow \langle \sigma'' \rangle}{\langle \text{int } x = i ; p, \sigma \rangle \Downarrow \langle \sigma'' \rangle} \quad \text{dacă } \sigma' = \sigma[i/x]$$

$$(\text{PGM}) \quad \frac{\langle p, \perp \rangle \Downarrow \langle \sigma \rangle}{\langle p \rangle \Downarrow \langle \sigma \rangle}$$

Arbore de derivare

$$\begin{array}{c}
 \text{(ID)} \quad \overline{\langle a, a \mapsto 3 \rangle \Downarrow \langle 3 \rangle} \qquad \text{(INT)} \quad \langle 4, a \mapsto 3 \rangle \Downarrow \langle 4 \rangle \\
 \text{(ADD)} \quad \frac{\overline{\langle a, a \mapsto 3 \rangle \Downarrow \langle 3 \rangle} \quad \langle 4, a \mapsto 3 \rangle \Downarrow \langle 4 \rangle}{\langle a + 4, a \mapsto 3 \rangle \Downarrow \langle 7 \rangle} \\
 \text{(ASGN)} \quad \frac{\langle a + 4, a \mapsto 3 \rangle \Downarrow \langle 7 \rangle}{\langle a = a + 4 ; , a \mapsto 3 \rangle \Downarrow \langle a \mapsto 7 \rangle} \\
 \text{(INIT)} \quad \frac{\langle a = a + 4 ; , a \mapsto 3 \rangle \Downarrow \langle a \mapsto 7 \rangle}{\langle \text{int } a = 3 ; a = a + 4 ; , \perp \rangle \Downarrow \langle a \mapsto 7 \rangle} \\
 \text{M)} \quad \frac{\langle \text{int } a = 3 ; a = a + 4 ; , \perp \rangle \Downarrow \langle a \mapsto 7 \rangle}{\langle \text{int } a = 3 ; a = a + 4 ; \rangle \Downarrow \langle a \mapsto 7 \rangle}
 \end{array}$$

Proprietăți ale limbajului IMP

- Intuitiv, două instrucțiuni sunt echivalente dacă produc același rezultat în orice stare de execuție a programului.
- Formal, două instrucțiuni c_1 și c_2 sunt **echivalente** (notat $c_1 \sim c_2$) dacă, pentru orice stări σ și σ'

$$\langle c_1, \sigma \rangle \Downarrow \langle \sigma' \rangle \quad \Leftrightarrow \quad \langle c_2, \sigma \rangle \Downarrow \langle \sigma' \rangle$$

Proprietăți ale limbajului IMP

- Intuitiv, două instrucțiuni sunt echivalente dacă produc același rezultat în orice stare de execuție a programului.
- Formal, două instrucțiuni c_1 și c_2 sunt **echivalente** (notat $c_1 \sim c_2$) dacă, pentru orice stări σ și σ'

$$\langle c_1, \sigma \rangle \Downarrow \langle \sigma' \rangle \Leftrightarrow \langle c_2, \sigma \rangle \Downarrow \langle \sigma' \rangle$$

De exemplu, putem demonstra următorul rezultat:

Teoremă

$$\text{while (b) } c \quad \sim \quad \text{if (b) } c ; \text{while (b) } c \text{ else } \{ \}$$

Teoremă (Determinism)

Dacă $\langle c, \sigma \rangle \Downarrow \langle \sigma' \rangle$ și $\langle c, \sigma \rangle \Downarrow \langle \sigma'' \rangle$ atunci $\sigma' = \sigma''$.