Fundamentele limbajelor de programare

C11

Denisa Diaconescu
Traian Șerbănuță

Departamentul de Informatică, FMI, UB

Semantica limbajelor de programare

Principalele paradigme de programare

- Imperativă (cum calculăm)
  - Procedurală
  - Orientată pe obiecte
- Declarativă (ce calculăm)
  - Logică
  - Functională

Fundamentele paradigmelor de programare

Imperativă Execuția unei Mașini Turing

Logică Rezoluția în logica clauzelor Horn

Funcțională Beta-reducție în Lambda Calcul

Ce înseamnă semantică formală?

Ce definește un limbaj de programare?

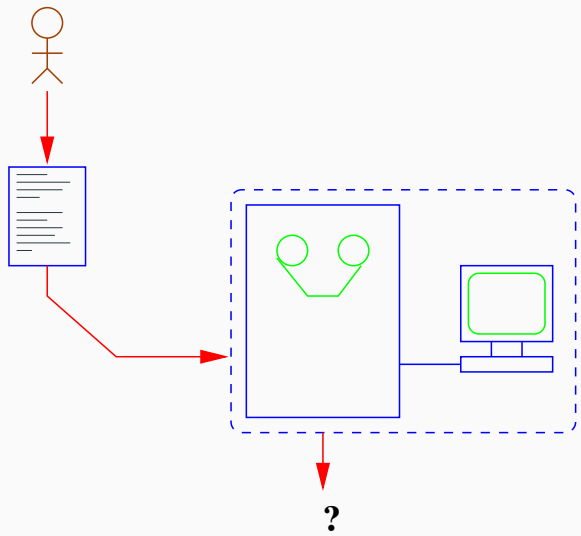
- Sintaxa – Simboluri de operație, cuvinte cheie, descriere (formală) a programelor/expresiilor bine formate
- Practic – Un limbaj e definit de modul cum poate fi folosit
  - Manual de utilizare și exemple de bune practici
  - Implementare (compilator/interpretor)
  - Instrumente ajutătoare (analizor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

La ce folosește semantica?

- Să înțelegem un limbaj în profunzime
  - Ca programator: pe ce mă pot baza când programez
  - 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

Problema corectitudinii programelor

- Pentru anumite metode de programare (e.g., imperativă, orientată pe obiecte), nu este ușor să stabilim dacă un program este corect sau să înțelegem ce înseamnă că este corect (e.g, în raport cu ce?!).
- Corectitudinea programelor devine o problemă din ce în ce mai importantă, nu doar pentru aplicații "safety-critical".
- Avem nevoie de metode ce asigură "calitate", capabile să ofere "garanții".



C

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

C

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

- Este corect?

C

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

- Este corect? În raport cu ce?

C

```
#include <iostream>
using namespace std;
int main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

- Este corect? În raport cu ce?
- Un formalism adecvat trebuie:
  - să permită descrierea problemelor (specificații), și
  - să raționeze despre implementarea lor (corectitudinea programelor).

Care este comportamentul corect?

```
int main(void) {
  int x = 0;
  return (x = 1) + (x = 2);
}
```

Care este comportamentul corect?

```
int main(void) {
  int x = 0;
  return (x = 1) + (x = 2);
}
```

- GCC4, MSVC: valoarea întoarsă e 4
- GCC3, ICC, Clang: valoarea întoarsă e 3

Conform standardului limbajului C (ISO/IEC 9899:2018)
Comportamentul programului este nedefinit.

Tipuri de semantică

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

Limbajul IMP

Vom folosi ca exemplu un mic limbaj imperativ IMP care conține:

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

Limbajul IMP

Un exemplu de 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

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

Semantica operațională small-step

Imagine de ansamblu

- Semantica operațională descrie cum se execută un program pe o masină abstractă (ideală).
- Semantica operațională **small-step**
  - semantica structurală, a pașilor mici
  - descrie cum o execuție a programului avansează în functie 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):

$$\sigma : \mathit{Var} \rightharpoonup \mathit{Int}$$

- Notatii:
  - Descrierea funcției prin enumerare: $\sigma = n \mapsto 10, \mathit{sum} \mapsto 0$
  - 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:

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

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

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

- Cum definim această relație?

  Prin inductie după elementele din sintaxă.

Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

$$\texttt{if}\ (0 \texttt{<=}\ 5 + 7 \texttt{*}\ x\ \texttt{,}\ r = 1\ \texttt{,}\ r = 0)$$

Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

$$\texttt{if}\ (0 \texttt{ <= } 5 \texttt{ + } 7 \texttt{ * } x \texttt{ , } r = 1 \texttt{ , } r = 0)$$

Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

$$\texttt{if}\ (0 \texttt{ <= } 5 \texttt{ + } 7 \texttt{ * } x \texttt{ , } r = 1 \texttt{ , } r = 0)$$

- 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

$$\texttt{if}\ (0 \texttt{ <= } 5 \texttt{ + } 7 \texttt{ * } x\ \texttt{ , } r = 1\ \texttt{ , } r = 0)$$

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

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

Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

$$\texttt{if}\ (0 \texttt{ <= } 5 \texttt{ + } 7 \texttt{ * } x\ \texttt{ , } r = 1\ \texttt{ , } r = 0)$$

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

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

- Axiome
  - Realizează pasul computațional

Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

$$\texttt{if}\ (0 \texttt{ <= } 5 \texttt{ + } 7 \texttt{ * } x\ \texttt{,}\ r = 1\ \texttt{,}\ r = 0)$$

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

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

- Axiome
  - Realizează pasul computațional

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

Semantica expresiilor aritmetice

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

  (Id) $\langle x\ ,\ \sigma\rangle \to \langle i\ ,\ \sigma\rangle \quad \textit{dacă}\ \sigma(x) = i$
- Semantica adunării a două expresii aritmetice

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

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

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

  Observatie: ordinea de evaluare a argumentelor este nespecificată.

Semantica expresiilor booleene

- Semantica operatorului de comparație

(LEQ-FALSE) $\langle i_1 =< i_2 \ , \ \sigma \rangle \to \langle \mathbf{false} \ , \ \sigma \rangle \quad$ *dacă* $i_1 > i_2$

(LEQ-TRUE) $\langle i_1 =< i_2 \ , \ \sigma \rangle \to \langle \mathbf{true} \ , \ \sigma \rangle \quad$ *dacă* $i_1 \leq i_2$

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

- Semantica negației

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

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

$$\frac{\langle a \ , \ \sigma \rangle \to \langle a' \ , \ \sigma \rangle}{\langle \mathbf{not} \ (a) \ , \ \sigma \rangle \to \langle \mathbf{not} \ (a') \ , \ \sigma \rangle}$$

Semantica expresiilor booleene

- Semantica și-ului

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

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

$$\frac{\langle b_1\ ,\ \sigma\rangle \to \langle b_1'\ ,\ \sigma\rangle}{\langle \mathbf{and}\ (b_1\ ,\ b_2)\ ,\ \sigma\rangle \to \langle \mathbf{and}\ (b_1'\ ,\ b_2)\ ,\ \sigma\rangle}$$

Semantica compunerii și a blocurilor

- Semantica blocurilor

  (Block) $\langle \texttt{\{ } s \texttt{ \}} , \sigma \rangle \to \langle s , \sigma \rangle$

- Semantica compunerii secvențiale

  (Next-stmt) $\langle \texttt{skip; } s_2 , \sigma \rangle \to \langle s_2 , \sigma \rangle$

  $$\frac{\langle s_1 , \sigma \rangle \to \langle s_1' , \sigma' \rangle}{\langle s_1 \texttt{ ; } s_2 , \sigma \rangle \to \langle s_1' \texttt{ ; } s_2 , \sigma' \rangle}$$

- Semantica atribuirii

  (Asgn) $\langle x = i , \sigma \rangle \to \langle \texttt{skip} , \sigma' \rangle \quad \textit{dacă } \sigma' = \sigma_{x \leftarrow i}$

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

Semantica lui `if`

- Semantica lui `if`

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

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

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

- Semantica lui `while`

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

- Semantica programelor

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

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

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } \bot \rangle \xrightarrow{\textsc{Asgn}}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } \bot \rangle \xrightarrow{\textsc{Asgn}}$$

$$\langle \texttt{skip; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{Next-stmt}}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } \bot \rangle \xrightarrow{\textsc{Asgn}}$$

$$\langle \texttt{skip; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{Next-stmt}}$$

$$\langle \texttt{while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{While}}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \,,\ \bot \rangle \xrightarrow{\textsc{Asgn}}$$

$$\langle \texttt{skip; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \,,\ i \mapsto 3 \rangle \xrightarrow{\textsc{Next-stmt}}$$

$$\langle \texttt{while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \,,\ i \mapsto 3 \rangle \xrightarrow{\textsc{While}}$$

$$\langle \texttt{if } (0 \texttt{ <= } i \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \,,\ i \mapsto 3 \rangle \xrightarrow{\textsc{Id}}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } \bot \rangle \xrightarrow{\textsc{Asgn}}$$

$$\langle \texttt{skip; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{Next-stmt}}$$

$$\langle \texttt{while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{While}}$$

$$\langle \texttt{if } (0 \texttt{ <= } i \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{Id}}$$

$$\langle \texttt{if } (0 \texttt{ <= } 3 \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3 \rangle \xrightarrow{\textsc{Leq-tru}}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } \bot\rangle \xrightarrow{\textsc{Asgn}}$$

$$\langle \texttt{skip; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Next-stmt}}$$

$$\langle \texttt{while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{While}}$$

$$\langle \texttt{if } (0 \texttt{ <= } i \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Id}}$$

$$\langle \texttt{if } (0 \texttt{ <= } 3 \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Leq-tru}}$$

$$\langle \texttt{if } (\texttt{true, } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{If-true}}$$

Semantica small-step a lui IMP

Execuție pas cu pas

$$\langle i = 3 \texttt{ ; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } \bot\rangle \xrightarrow{\textsc{Asgn}}$$

$$\langle \texttt{skip; while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Next-stmt}}$$

$$\langle \texttt{while } (0 \texttt{ <= } i \texttt{ ,\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{While}}$$

$$\langle \texttt{if } (0 \texttt{ <= } i \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Id}}$$

$$\langle \texttt{if } (0 \texttt{ <= } 3 \texttt{ , } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Leq-tru}}$$

$$\langle \texttt{if } (\texttt{true, } i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ ,skip}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{If-true}}$$

$$\langle i = i + -4 \texttt{ ;while } (0 \texttt{ <= } i, \texttt{\{ } i = i + -4 \texttt{ \}}) \texttt{ , } i \mapsto 3\rangle \xrightarrow{\textsc{Id}}$$

$\ldots$

Semantica axiomatică

Semantica Axiomatică

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

$$\{P\}\ \mathbb{C}\ \{Q\}$$

  unde:
  - $\mathbb{C}$ este o instrucțiune
  - P (precondiție), Q (postcondiție) sunt aserțiuni logice asupra stării sistemului înaintea, respectiv după execuția lui $\mathbb{C}$
  - Limbajul aserțiunilor este un limbaj de ordinul I.

Semantica Axiomatică

Interpretarea unui triplet Hoare $\{P\}\ \mathbb{C}\ \{Q\}$

- dacă programul se execută dintr-o stare inițială care satisface P
- și execuția se termină
- atunci se ajunge într-o stare finală care satisface Q.

Semantica Axiomatică

Interpretarea unui triplet Hoare $\{P\}\ \mathbb{C}\ \{Q\}$

- dacă programul se execută dintr-o stare inițială care satisface P
- și execuția se termină
- atunci se ajunge într-o stare finală care satisface Q.

Exemple:

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

Logica Hoare ne ajută să verificăm corectitudinea programelor

- Se asociază fiecărei construcții sintactice o regulă de deducție care definește recursiv tripletele corecte pentru un limbaj.
- Se exprimă o aserțiune de corectitudine a programului ca un tripet Hoare
- Se verifică dacă tripletul dat e corect folosind definiția recursivă

Sistem de reguli pentru logica Floyd-Hoare

Reguli generale pentru logică propozițională

$$(\rightarrow) \quad \frac{P1 \rightarrow P2 \quad \{P2\}\ \mathbb{C}\ \{Q2\} \quad Q2 \rightarrow Q1}{\{P1\}\ \mathbb{C}\ \{Q1\}}$$

$$(\vee) \quad \frac{\{P1\}\ \mathbb{C}\ \{Q\} \quad \{P2\}\ \mathbb{C}\ \{Q\}}{\{P1 \vee P2\}\ \mathbb{C}\ \{Q\}}$$

$$(\wedge) \quad \frac{\{P\}\ \mathbb{C}\ \{Q1\} \quad \{P\}\ \mathbb{C}\ \{Q2\}}{\{P\}\ \mathbb{C}\ \{Q1 \wedge Q2\}}$$

Logica Floyd-Hoare pentru IMP1

$$(\textsc{Skip}) \quad \frac{}{\{P\}\ \{\}\ \{P\}}$$

$$(\textsc{Seq}) \quad \frac{\{P\}\ \mathbb{C}_1\ \{Q\} \quad \{Q\}\ \mathbb{C}_2\ \{R\}}{\{P\}\ \mathbb{C}_1 ; \mathbb{C}_2\ \{R\}}$$

$$(\textsc{Asign}) \quad \frac{}{\{P[x/e]\}\ x = e\ \{P\}}$$

$$(\textsc{If}) \quad \frac{\{\mathbb{B} \wedge P\}\ \mathbb{C}_1\ \{Q\} \quad \{\neg\mathbb{B} \wedge P\}\ \mathbb{C}_2\ \{Q\}}{\{P\}\ \texttt{if}\ (\mathbb{B})\ \mathbb{C}_1\ \texttt{else}\ \mathbb{C}_2\ \{Q\}}$$

$$(\textsc{While}) \quad \frac{\{\mathbb{B} \wedge P\}\ \mathbb{C}\ \{P\}}{\{P\}\ \texttt{while}\ (\mathbb{B})\ \mathbb{C}\ \{\neg\mathbb{B} \wedge P\}}$$

Regula pentru atribuire

$$(\textsc{Asign}) \quad \frac{}{\{P[x/e]\}\ x = e\ \{P\}}$$

Exemplu:

$$\{\mathtt{x} + \mathtt{y} = \mathtt{y} + 10\}\ \mathtt{x}\ =\ \mathtt{x}\ +\ \mathtt{y}\ \{\mathtt{x} = \mathtt{y} + 10\}$$

Regula pentru condiții

$$(\textsc{If}) \quad \frac{\{\mathbb{B} \wedge P\}\ \mathbb{C}_1\ \{Q\} \quad \{\neg\mathbb{B} \wedge P\}\ \mathbb{C}_2\ \{Q\}}{\{P\}\ \texttt{if}\ (\mathbb{B})\ \mathbb{C}_1\ \texttt{else}\ \mathbb{C}_2\ \{Q\}}$$

Exemplu:

Pentru a demonstra

$\{\top\}\ \texttt{if (x <= y) z = x; else z = y;}\ \{\texttt{z} = \mathit{min}(\texttt{x}, \texttt{y})\}$

este suficient să demonstrăm

- $\{\texttt{x} \leq \texttt{y}\}\ \texttt{z = x}\ \{\texttt{z} = \mathit{min}(\texttt{x}, \texttt{y})\}$
- $\{\neg(\texttt{x} \leq \texttt{y})\}\ \texttt{z = y}\ \{\texttt{z} = \mathit{min}(\texttt{x}, \texttt{y})\}$

Invarianți pentru `while`

Cum demonstrăm $\{P\}$ `while(`$\mathbb{B}$`)` $\mathbb{C}$ $\{Q\}$?

Se determină un invariant I și se folosește următoarea regulă:

$$(\textsc{Inv}) \quad \frac{P \to I \quad \{\mathbb{B} \wedge I\}\,\mathbb{C}\,\{I\} \quad (I \wedge \neg\mathbb{B}) \to Q}{\{P\}\ \mathbf{while}\ (\mathbb{B})\,\mathbb{C}\,\{Q\}}$$

Invariantul trebuie să satisfacă următoarele proprietăți:

- să fie adevărat inițial
- să rămână adevărat după execuția unui ciclu
- să implice postcondiția la ieșirea din buclă

Invarianți pentru `while`

$\{\texttt{x} = 0 \ \wedge \ 0 \leq n \ \wedge \texttt{y} = 1\}$

```
while (x < n) { x = x + 1; y = y * x}
```

$\{\texttt{y} = n!\}$

Invarianți pentru `while`

$\{\texttt{x} = 0 \ \wedge \ 0 \leq n \ \wedge \texttt{y} = 1\}$

```
while (x < n) { x = x + 1; y = y * x}
```

$\{\texttt{y} = n!\}$

- Invariantul I este $\texttt{y} = x!$

Invarianți pentru `while`

$\{\mathtt{x} = 0 \ \wedge \ 0 \leq n \ \wedge \mathtt{y} = 1\}$

```
while (x < n) { x = x + 1; y = y * x}
```

$\{\mathtt{y} = n!\}$

- Invariantul I este $\mathtt{y} = x!$
- $(\mathtt{x} = 0 \ \wedge \ 0 \leq n \ \wedge \mathtt{y} = 1) \rightarrow I$
- $\{I \wedge (x < n)\}$ `x = x + 1; y = y * x` $\{I\}$
- $I \wedge \neg(\mathtt{x} < n) \rightarrow (\mathtt{y} = n!)$

Pe data viitoare!