Curs 8-9

2021-2022

Fundamentele limbajelor de programare

Cuprins

Limbajul IMP

Limbajul IMP

Vom implementa un limbaj care conţine:

- Expresii
  - Aritmetice — `x + 3`
  - Booleene — `x >= 7`
- Instructiuni
  - De atribuire — `x = 5`
  - Conditionale — `if(x >= 7, x =5, x = 0)`
  - De ciclare — `while(x >= 7,x = x - 1)`
- Compunerea instrutiunilor — `x=7;while(x>=0,x=x-1)`
- Blocuri de instrucţiuni — `{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

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

Semantica programelor - idei generale

Ce înseamnă semantica formală?

Ce defineşte un limbaj de programare?

Ce înseamnă semantica formală?

Ce defineşte un limbaj de programare?

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

Ce înseamnă semantica formală?

Ce defineşte un limbaj de programare?

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

Ce înseamnă semantica formală?

Ce defineşte un limbaj de programare?

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

La ce foloseşte semantica?

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

La ce foloseşte semantica?

- Să înţelegem un limbaj în profunzime
  - Ca programator: pe ce mă pot baza când programez în limbajul dat
  - Ca implementator al limbajului: ce garanţii trebuie să ofer
- Ca instrument în proiectarea unui nou limbaj/a unei extensii
  - Înţelegerea componentelor şi a relaţiilor dintre ele
  - Exprimarea (şi motivarea) deciziilor de proiectare
  - Demonstrarea unor proprietăţi generice ale limbajului

La ce foloseşte semantica?

- Să înţelegem un limbaj în profunzime
  - Ca programator: pe ce mă pot baza când programez în limbajul dat
  - Ca implementator al limbajului: ce garanţii trebuie să ofer
- Ca instrument în proiectarea unui nou limbaj/a unei extensii
  - Înţelegerea componentelor şi a relaţiilor dintre ele
  - Exprimarea (şi motivarea) deciziilor de proiectare
  - Demonstrarea unor proprietăţi 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 instrucţiuni
  - $\vdash \{\varphi\} cod \{\psi\}$
  - modelează un program prin formulele logice pe care le satisface
  - utilă pentru demonstrarea corectitunii

Tipuri de semantică

- Limbaj natural – descriere textuală a efectelor
- Axiomatică – descrierea folosind logică a efectelor unei instrucţiuni
  - $\vdash \{\varphi\} cod \{\psi\}$
  - modelează un program prin formulele logice pe care le satisface
  - utilă pentru demonstrarea corectitunii
- Denotaţională – asocierea unui obiect matematic (denotaţie)
  - $[\![cod]\!]$
  - 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 instrucţiuni
  - $\vdash \{\varphi\} cod \{\psi\}$
  - modelează un program prin formulele logice pe care le satisface
  - utilă pentru demonstrarea corectitunii
- Denotaţională – asocierea unui obiect matematic (denotaţie)
  - $[\![cod]\!]$
  - modelează un program ca obiecte matematice
  - utilă pentru fundamente matematice
- Operaţională – asocierea unei demonstraţii pentru execuţie
  - $\langle cod, \sigma \rangle \to \langle cod', \sigma' \rangle$
  - modelează un program prin execuţia pe o maşină abstractă
  - utilă pentru implementarea de compilatoare şi interpretoare

Tipuri de semantică

- Limbaj natural – descriere textuală a efectelor
- Axiomatică – descrierea folosind logică a efectelor unei instrucţiuni
  - $\vdash \{\varphi\} cod \{\psi\}$
  - modelează un program prin formulele logice pe care le satisface
  - utilă pentru demonstrarea corectitunii
- Denotaţională – asocierea unui obiect matematic (denotaţie)
  - $[\![cod]\!]$
  - modelează un program ca obiecte matematice
  - utilă pentru fundamente matematice
- Operaţională – asocierea unei demonstraţii pentru execuţie
  - $\langle cod, \sigma \rangle \to \langle cod', \sigma' \rangle$
  - modelează un program prin execuţia pe o maşină abstractă
  - utilă pentru implementarea de compilatoare şi interpretoare
- Statică – asocierea unui sistem de tipuri care exclude programe eronate

Semantica denotaţională

Semantica denotațională

- Introdusă de Christopher Strachey şi Dana Scott (1970)
- Semantica operaţională, ca un interpretor, descrie cum să evaluăm un program.
- Semantica denotaţională, ca un compilator, descrie o traducere a limbajului într-un limbaj diferit cu semantică cunoscută, anume matematica.
- Semantica denotaţională defineşte ce înseamnă un program ca o funcţie matematică.

Semantica denotațională

- Definim stările memoriei ca fiind funcţii parţiale de la mulţimea identificatorilor la mulţimea valorilor:

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

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

Semantica denotațională

- Definim stările memoriei ca fiind funcţii parţiale de la mulţimea identificatorilor la mulţimea valorilor:

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

- Asociem fiecărei categorii sintactice o categorie semantică.
- Fiecare construcţie sintactică va avea o denotaţie (interpretare) în categoria semantică respectivă.De exemplu:
  - denotaţia unei expresii aritmetice este o funcţie parţială de la mulţimea stărilor memoriei la mulţimea valorilor ($\mathbb{Z}$):

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

  - denotaţia unei instrucţiuni este o funcţie parţială de la mulţimea stărilor memoriei la mulţimea stărilor memoriei:

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

Semantica denotațională

$State = Id \rightharpoonup \mathbb{Z}$
$[\![\_]\!] : AExp \to (State \rightharpoonup \mathbb{Z})$
$[\![\_]\!] : Stmt \to (State \rightharpoonup State)$

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.

Semantica denotațională

$State = Id \rightharpoonup \mathbb{Z}$
$[[\_]] : AExp \to (State \rightharpoonup \mathbb{Z})$
$[[\_]] : Stmt \to (State \rightharpoonup State)$

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: expresii

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

- Domenii semantice:
  $[\![\_]\!] : AExp \to (State \rightharpoonup \mathbb{Z})$
  $[\![\_]\!] : BExp \to (State \rightharpoonup \{T, F\})$
  $[\![\_]\!] : Stmt \to (State \rightharpoonup State)$

Semantica denotațională: expresii

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

- Domenii semantice:
  $[[\_]] : AExp \to (State \rightharpoonup \mathbb{Z})$
  $[[\_]] : BExp \to (State \rightharpoonup \{T, F\})$
  $[[\_]] : Stmt \to (State \rightharpoonup State)$
- Semantica denotaţională este compoziţională:
  - semantica expresiilor aritmetice
    $[[n]](s) = n$
    $[[x]](s) = s(x)$
    $[[e1 + e2]](s) = [[e1]](s) + [[e2]](s)$

Semantica denotațională: expresii

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

- Domenii semantice:
  $[[\_]] : AExp \to (State \rightharpoonup \mathbb{Z})$
  $[[\_]] : BExp \to (State \rightharpoonup \{T, F\})$
  $[[\_]] : Stmt \to (State \rightharpoonup State)$
- Semantica denotaţională este compoziţională:
  - 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 denotaţională: instrucţiuni

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

- Domenii semantice:
  $[\![\_]\!] : AExp \to (State \rightharpoonup \mathbb{Z})$
  $[\![\_]\!] : BExp \to (State \rightharpoonup \{T, F\})$
  $[\![\_]\!] : Stmt \to (State \rightharpoonup State)$

Semantica denotaţională: instrucţiuni

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

- Domenii semantice:
  $[\![\_]\!] : AExp \to (State \rightharpoonup \mathbb{Z})$
  $[\![\_]\!] : BExp \to (State \rightharpoonup \{T, F\})$
  $[\![\_]\!] : Stmt \to (State \rightharpoonup State)$
- Semantica instrucţiunilor:

$$[\![\texttt{skip}]\!] = id$$

$$[\![\texttt{c1;c2}]\!] = [\![c2]\!] \circ [\![c1]\!]$$

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

Semantica denotaţională: instrucţiuni

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

- Domenii semantice:
  $[[\_]] : AExp \to (State \rightharpoonup \mathbb{Z})$
  $[[\_]] : BExp \to (State \rightharpoonup \{T, F\})$
  $[[\_]] : Stmt \to (State \rightharpoonup State)$
- Semantica instrucţiunilor:

$$[[\texttt{skip}]] = id$$

$$[[\texttt{c1;c2}]] = [[c2]] \circ [[c1]]$$

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

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

Semantica denotațională

Exemplu

```
if (x<= y) z=x; else z=y;
```

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

Semantica denotațională

Exemplu

```
if (x<= y) z=x; else z=y;
```

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

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

Semantica denotațională

Exemplu

```
if (x<= y) z=x; else z=y;
```

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

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

Cum definim semantica denotaţională pentru `while`?

Mulţimea funcţiilor parţiale

Fie X şi Y două mulţimi.

- $Pfn(X, Y)$ mulţimea funcţiilor parţiale de la X la Y, adică $Pfn(X, Y) = X \rightharpoonup Y$
- Pentru $f \in Pfn(X, Y)$ notăm cu $dom(f)$ mulţimea elementelor din X pentru care funcţia este definită.
  Atunci $dom(f) \subseteq X$ şi $f|_{dom(f)} : dom(f) \to Y$ este funcţie.

Mulţimea funcţiilor parţiale

Fie X şi Y două mulţimi.

- $Pfn(X, Y)$ mulţimea funcţiilor parţiale de la X la Y, adică $Pfn(X, Y) = X \rightharpoonup Y$
- Pentru $f \in Pfn(X, Y)$ notăm cu $dom(f)$ mulţimea elementelor din X pentru care funcţia este definită.
  Atunci $dom(f) \subseteq X$ şi $f|_{dom(f)} : dom(f) \to Y$ este funcţie.
- Fie $\perp : X \rightharpoonup Y$ unica funcţie cu $dom(\perp) = \emptyset$ (funcţia care nu este definită în nici un punct).
- Definim pe $Pfn(X, Y)$ următoarea relaţie:

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

Mulţimea funcţiilor parţiale

Fie X şi Y două mulţimi.

- $Pfn(X, Y)$ mulţimea funcţiilor parţiale de la X la Y, adică $Pfn(X, Y) = X \rightharpoonup Y$
- Pentru $f \in Pfn(X, Y)$ notăm cu $dom(f)$ mulţimea elementelor din X pentru care funcţia este definită.
  Atunci $dom(f) \subseteq X$ şi $f|_{dom(f)} : dom(f) \to Y$ este funcţie.
- Fie $\perp : X \rightharpoonup Y$ unica funcţie cu $dom(\perp) = \emptyset$ (funcţia care nu este definită în nici un punct).
- Definim pe $Pfn(X, Y)$ următoarea relaţie:

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

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

(mulţime parţial ordonată completă în care $\perp$ este cel mai mic element)

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

Exemplu

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

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

- $\mathbf{F}$ este o funcţie continuă,

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

Exemplu

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

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

- $\mathbf{F}$ este o funcţie continuă,deci putem aplica
- Teorema Knaster-Tarski
  Fie $g_n = \mathbf{F}^n(\perp)$ şi $f = \bigvee_n g_n$.
  Ştim că f este cel mai mic punct fix al funcţiei $\mathbf{F}$, deci $\mathbf{F}(f) = f$.

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

Exemplu

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

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

- $\mathbf{F}$ este o funcţie continuă,deci putem aplica
- Teorema Knaster-Tarski
  Fie $g_n = \mathbf{F}^n(\perp)$ şi $f = \bigvee_n g_n$.
  Ştim că f este cel mai mic punct fix al funcţiei $\mathbf{F}$, deci $\mathbf{F}(f) = f$.
- Demonstrăm prin inducţie după n că:

  $dom(g_n) = \{0, \ldots, n\}$ şi $g_n(k) = k!$ oricare $k \in dom(g_n)$
- $f : \mathbb{N} \to \mathbb{N}$ este funcţia factorial.

Semantica denotaţională pentru `while`

`while (b) c`

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

Semantica denotaţională pentru `while`

`while (b) c`

- Definim $\mathbf{F} : \mathit{Pfn}(\mathit{State}, \mathit{State}) \to \mathit{Pfn}(\mathit{State}, \mathit{State})$ 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: $\mathit{fix}(\mathbf{F}) = \bigcup_n \mathbf{F}^n(\bot)$

Semantica denotaţională pentru `while`

`while (b) c`

- Definim $\mathbf{F} : \mathit{Pfn}(\mathit{State}, \mathit{State}) \to \mathit{Pfn}(\mathit{State}, \mathit{State})$ 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: $\mathit{fix}(\mathbf{F}) = \bigcup_n \mathbf{F}^n(\bot)$
- Semantica denotaţională:

$$[[\_]] : \mathit{Stmt} \to (\mathit{State} \rightharpoonup \mathit{State})$$

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

Avantaje şi dezavantaje

Semantica operațională

+ Defineşte precis noțiunea de pas computațional
+ Semnalează erorile, oprind execuția
+ Execuția devine uşor de urmărit şi depanat
– Regulile structurale sunt evidente şi deci plictisitor de scris
– Nemodular: adăugarea unei trăsături noi poate solicita schimbarea întregii definiții

Semantica denotațională

+ Formală, matematică, foarte precisă
+ Compoziţională (morfisme şi compuneri de funcţii)
– Domeniile devin din ce în ce mai complexe.

Semantica axiomatică

Semantica Axiomatică

- Inventată de 1969 Tony Hoare în 1969 (insiprată de rezultatele lui Robert Floyd).
- Defineşte 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 aserţiunilor este un limbaj de ordinul I.

- Tripletul $\{Pre\}\ S\ \{Post\}$ este (parţial) corect dacă:
  - dacă programul se execută dintr-o stare iniţială care satisface *Pre*
  - şi execuţia se termină
  - atunci se ajunge într-o stare finală care satisface *Post*.

Semantica Axiomatică

Defineşte triplete (triplete Hoare) de forma

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

- Tripletul $\{Pre\}\ S\ \{Post\}$ este (parţial) corect dacă:
  - dacă programul se execută dintr-o stare iniţială care satisface *Pre*
  - şi execuţia se termină
  - atunci se ajunge într-o stare finală care satisface *Post*.

Exemplu

- $\{\mathtt{x} = 1\}$ `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

Semantica Axiomatică

Defineşte triplete (triplete Hoare) de forma

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

unde:

- S este o instrucțiune (Stmt)
- *Pre* (precondiție), respectiv *Post* (postcondiție) sunt asertiuni logice asupra stării sistemului înaintea, respectiv după execuția lui S

Se asociază fiecărei construcții sintactice Stmt o regulă de deducție care defineşte recursiv tripletele Hoare descrise mai sus.

Sistem de reguli pentru logica Floyd-Hoare

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

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

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

Logica Floyd-Hoare pentru IMP1

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

$$(\textsc{Seq}) \quad \frac{\{P\}\ c1\ \{Q\} \quad \{Q\}\ c2\ \{R\}}{\{P\}\ c1; c2\ \{R\}}$$

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

$$(\textsc{If}) \quad \frac{\{b \wedge P\}\ c1\ \{Q\} \quad \{\neg b \wedge P\}\ c2\ \{Q\}}{\{P\}\ \texttt{if}\ (b) c1\ \texttt{else}\ c2\ \{Q\}}$$

$$(\textsc{While}) \quad \frac{\{b \wedge P\}\ c\ \{P\}}{\{P\}\ \texttt{while}\ (b)\ c\ \{\neg b \wedge P\}}$$

Logica Floyd-Hoare pentru IMP1

- regula pentru atribuire

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

Exemplu

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

Logica Floyd-Hoare pentru IMP1

- 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 condiţii

$$(\textsc{If}) \quad \frac{\{b \wedge P\}\ c1\ \{Q\} \quad \{\neg b \wedge P\}\ c2\ \{Q\}}{\{P\}\ \mathtt{if}\ (b) c1\ \mathtt{else}\ c2\ \{Q\}}$$

Exemplu

Pentru a demonstra $\{\top\}$ `if (x<=y) z=x; else z=y;` $\{\mathtt{z} = \mathit{min}(\mathtt{x}, \mathtt{y})\}$
este suficient să demonstrăm $\{\mathtt{x} \leq \mathtt{y}\}$ `z=x;` $\{\mathtt{z} = \mathit{min}(\mathtt{x}, \mathtt{y})\}$
şi $\{\neg(\mathtt{x} \leq \mathtt{y})\}$ `z=y;` $\{\mathtt{z} = \mathit{min}(\mathtt{x}, \mathtt{y})\}$

Invarianţi pentru `while`

Cum demonstrăm $\{P\}$ `while (b)c` $\{Q\}$?

- Se determină un invariant I şi se foloseşte următoarea regulă:

$$(\textsc{Inv}) \quad \frac{P \to I \quad \{b \wedge I\}\ c\ \{I\} \quad (I \wedge \neg b) \to Q}{\{P\}\ \texttt{while}\ (b)\ c\ \{Q\}}$$

Invarianţi pentru `while`

Cum demonstrăm $\{P\}$ `while (b)c` $\{Q\}$?

- Se determină un invariant I şi se foloseşte următoarea regulă:

$$(\textsc{Inv}) \quad \frac{P \to I \quad \{b \wedge I\}\ c\ \{I\} \quad (I \wedge \neg b) \to Q}{\{P\}\ \texttt{while}\ (b)\ c\ \{Q\}}$$

Invariantul trebuie să satisfacăurmătoarele proprietăţi:

- să fie adevărat iniţial
- să rămână adevărat după executarea unui ciclu
- să implice postcondiţia la ieşirea din buclă

Invarianţi pentru `while`

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

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

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

Invarianţi 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 $\mathtt{y} = x!$

Invarianţi 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;` $\{I\}$
- $I \wedge \neg(\mathtt{x} < n) \rightarrow (\mathtt{y} = n!)$

Dafny

- Dezvoltat la Microsoft Research
- Un limbaj imperativ compilat open-source
- Suportă demonstraţii formale folosind precondiţii, postcondiţii, invarianţi de bucle
- Demonstrează şi terminarea programelor
- Concepte din diferite paradigme de programare
  - Programare imperativă: `if`, `while`, `:=`, ...
  - Programare funcţională: `function`, `datatype`, ...
  - ...

Dafny

- Pagina limbajului Dafny

  `https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/`

- Pagina de Github

  `https://github.com/dafny-lang/dafny`

- Dafny in browser

  `http://cse-212294.cse.chalmers.se/courses/tdv/dafny/`

- Tutorial

  `https://dafny-lang.github.io/dafny/OnlineTutorial/guide`

Dafny - Hello World

Dafny - fără erori

```
method Main() {
  print "hello, Dafny";
  assert 2 < 10;
}
```

Dafny - Hello World

Dafny - fără erori

```
method Main() {
  print "hello, Dafny";
  assert 2 < 10;
}
```

Dafny - erori

```
method Main() {
  print "hello, Dafny";
  assert 10 < 2;
}
```

Dafny - maximul a două numere

Următoarea metodă calculează maximul a doi întregi:

Dafny

```
method max (x : int, y : int) returns (z : int)
{
 if (x <= y) { return y; }
 return x;
}
```

Dar cum verificăm formal acest lucru?

Dafny - maximul a două numere

Adăugăm postcondiţii!

Dafny

```
method max (x : int, y : int) returns (z : int)
ensures (x <= z) && (y <= z)
{
 if (x <= y) { return y; }
 return x;
}
```

Dafny - maximul a două numere

Adăugăm postcondiţii!

Dafny

```
method max (x : int, y : int) returns (z : int)
ensures (x <= z) && (y <= z)
{
 if (x <= y) { return y; }
 return x;
}
```

Dar este de ajuns condiţia de mai sus?

Dafny - maximul a două numere

Adăugăm postcondiţii!

Dafny

```
method max (x : int, y : int) returns (z : int)
ensures (x <= z) && (y <= z)
{
 if (x <= y) { return y; }
 return x;
}
```

Dar este de ajuns condiţia de mai sus?
O metodă este reprezentată prin precondiţiile şi postcondiţiile pe care le satisface, iar codul este "ignorat" ulterior.
În exemplul de mai sus, pentru `x = 3` şi `y = 6`, `z = 7` satisface postcondiţia (dacă ignorăm codul) dar nu este maximul dintre `x` şi `y`.

Dafny - maximul a două numere

Dafny

```
method max (x : int, y : int) returns (z : int)
ensures (x <= z) && (y <= z)
ensures (x == z) || (y == z)
{
 if (x <= y) { return y; }
 return x;
}
```

Dafny - suma primelor n numere impare

Cum demonstram formal că suma primelor n numere impare este n^2?

- $1 = 1 = 1^2$
- $1 + 3 = 4 = 2^2$
- $1 + 3 + 5 = 9 = 3^2$
- $1 + 3 + 5 + 7 = 16 = 4^2$

Dafny - suma primelor n numere impare

Cum demonstram formal că suma primelor n numere impare este n^2?

Dafny

```
method oddSum(n: int) returns (s: int)
{
 var i: int := 0;
 s := 0;
 while i != n
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Dafny - suma primelor n numere impare

Adăugăm postcondiţia!

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Dar Dafny nu reuşeşte să o demonstreze!

Dafny - suma primelor n numere impare

Adăugăm postcondiţia!

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Dar Dafny nu reuşeşte să o demonstreze! Trebuie să ii spunem ce se întâmplă în buclă prin invarianţi.

Dafny - suma primelor n numere impare

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 invariant s == i*i
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Dafny - suma primelor n numere impare

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 invariant s == i*i
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Acum Dafny se plânge că nu reuşeşte să demonstreze terminarea!
Adăugăm un nou invariant care ne asigură că *i* nu depaşeşte *n*.

Dafny - suma primelor n numere impare

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 invariant 0 <= i <= n
 invariant s == i*i
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Dafny - suma primelor n numere impare

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 invariant 0 <= i <= n
 invariant s == i*i
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Totuşi Dafny nu reuşeşte să demonstreze postcondiţia! De ce?

Dafny - suma primelor n numere impare

Dafny

```
method oddSum(n: int) returns (s: int)
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 invariant 0 <= i <= n
 invariant s == i*i
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Totuşi Dafny nu reuşeşte să demonstreze postcondiţia! De ce?
Ce se întâmplă dacă $n < 0$?

Dafny - suma primelor n numere impare

Dafny

```
method oddSum(n: int) returns (s: int)
requires 0 <= n
ensures s == n * n
{
 var i: int := 0;
 s := 0;
 while i != n
 invariant 0 <= i <= n
 invariant s == i*i
 {
  i := i + 1;
  s := s + (2*i - 1);
 }
}
```

Dafny - funcţia factorial

Dafny

```
function factorial(n: nat): nat
{ if n==0 then 1 else n*factorial(n-1) }

method CheckFactorial(n: nat) returns (r: nat)
  ensures r == factorial(n)
{
  var i := 0;
  r := 1;
  while i < n
    invariant r == factorial(i)
    invariant 0 <= i <= n
   {
    i := i + 1;
    r := r * i;
    }}
```

Pe săptămâna viitoare!