Curs 1

2020-2021

Fundamentele Limbajelor de Programare

Cuprins

Organizare

Instructori

Curs:

- **Ioana Leuștean (seria 24), Traian-Florin Șerbănuță (seria 23)**

Laborator

Seria 24
- **Ioana Leuștean (241, 244)**
- **Natalia Ozunu (242, 243)**

Seria 23
- **Ana Țurlea (231, 232, 233)**
- **Traian Șerbănuță (234)**

Suport curs/seminar/laborator

Seria 24
- https://cs.unibuc.ro//~ileustean/FLP.html
- Moodle: https://moodle.unibuc.ro/course/view.php?id=4635
- Materiale Curs/Laborator: https://bit.ly/3de0SOF

Seria 23
- Materiale Curs/Laborator: http://bit.do/unibuc-flp
- Moodle (teste, note): https://moodle.unibuc.ro/course/view.php?id=4634

O parte din materiale sunt realizate în colaborare cu Denisa Diaconescu.

Notare

- **Testare parțială: 40 puncte**
- **Testare finală: 50 puncte**
- Se acordă 10 puncte din oficiu!

Notare

- **Testare parțială: 40 puncte**
- **Testare finală: 50 puncte**
- Se acordă 10 puncte din oficiu!

- Condiţie minimă pentru promovare:
  testare parțială: minim 20 puncte şi
  testare finală: minim 20 puncte.

Notare

- **Testare parțială: 40 puncte**
- **Testare finală: 50 puncte**
- Se acordă 10 puncte din oficiu!

- Condiţie minimă pentru promovare:
  testare parțială: minim 20 puncte şi
  testare finală: minim 20 puncte.

- Se poate obţine punctaj suplimentar pentru activitatea din timpul laboratorului:
  maxim 10 puncte.

Testare parțială: 40 puncte

- Data: 23 aprilie
- Timp de lucru: 1,5 ore
- Prezenţa este obligatorie pentru a putea promova!
- Pentru a trece această probă, trebuie să obţineţi minim 20 de puncte.

Testare finală: 50 puncte

- Data: În sesiune
- Timp de lucru: 2 ore
- Prezenţa este obligatorie pentru a putea promova!
- Pentru a trece această probă, trebuie să obţineţi minim 20 de puncte.

Curs/seminar/laborator

- Curs

  Semantica limbajelor de programare
  - Parsare, Verificarea tipurilor și Interpretare
  - Semantică operațională, statică și axiomatică
  - Inferarea automată a tipurilor

  Bazele programării funcționale
  - Lambda Calcul, Codificări Church, combinatori
  - Lambda Calcul cu tipuri de date

  Bazele programării logice
  - Logica clauzelor Horn, Unificare, Rezoluţie

- Laborator:

  Haskell Limbaj pur de programare funcțională
  - Interpretoare pentru mini-limbaje

  Prolog Cel mai cunoscut limbaj de programare logică
  - Verificator pentru un mini-limbaj imperativ
  - Inferența tipurilor pentru un mini-limbaj funcțional

Bibliografie

- B.C. Pierce, **Types and programming languages**. MIT Press.2002
- G. Winskel, **The formal semantics of programming languages**. MIT Press. 1993
- H. Barendregt, E. Barendsen, **Introduction to Lambda Calculus**, 2000.
- J. Lloyd. **Foundations of Logic Programming**, second edition. Springer, 1987.
- P. Blackburn, J. Bos, and K. Striegnitz, **Learn Prolog Now!** (Texts in Computing, Vol. 7),College Publications, 2006
- M. Huth, M. Ryan, **Logic in Computer Science (Modelling and Reasoning about Systems)**, Cambridge University Press, 2004.

Privire de ansamblu

Semantica Limbajelor de Programare

Ce definește un limbaj de programare?

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

Practica 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, verificator de tipuri, depanator)

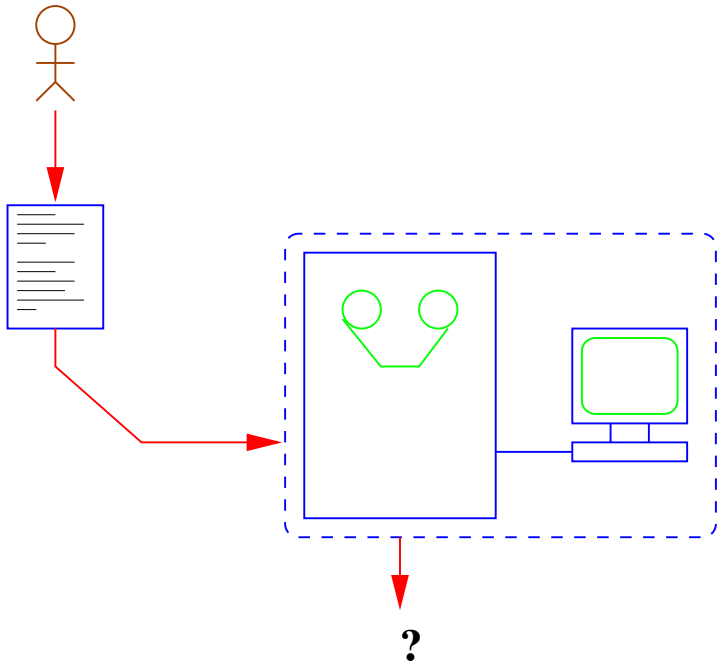
Semantica? Ce înseamnă / care e comportamentul unei instrucțiuni?

- 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 în limbajul dat
  - Ca implementator al limbajului: ce garanții trebuie să ofer
- Ca instrument în proiectarea unui nou limbaj / a unei extensii
  - Înțelegerea componentelor și a relațiilor dintre ele
  - Exprimarea (și motivarea) deciziilor de proiectare
  - Demonstrarea unor proprietăți generice ale limbajului
    E.g., execuția nu se va bloca pentru programe care trec de analiza tipurilor
- Ca bază pentru demonstrarea corectitudinii programelor.

Problema corectitudinii programelor



- Pentru anumite metode de programare (e.g., imperativă, orientată pe obiecte), nu este uşor să stabilim că un program este corect sau să înţelegem 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 aplicaţii "safety-critical".
- Avem nevoie de metode ce asigură "calitate", capabile să ofere "garanţii".

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 (specificaţii), şi
  - să raţioneze 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);
}
```

Conform standardului C, comportamentul programului este nedefinit.

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

Care este comportamentul corect?

```
int r;
int f(int x) {
  return (r = x);
}
int main() {
  return f(1) + f(2), r;
}
```

Care este comportamentul corect?

```
int r;
int f(int x) {
  return (r = x);
}
int main() {
  return f(1) + f(2), r;
}
```

Conform standardului C, comportamentul programului este corect, dar subspecificat:
poate întoarce atât valoarea 1 cât și 2.

Tipuri de semantică

Semantica dă "înţeles" unui program.

Tipuri de semantică

Semantica dă "înţeles" unui program.

- Operaţională:
  - Înţelesul programului este definit în funcţie de paşii (transformări dintr-o stare în alta) care apar în timpul execuţiei.

Tipuri de semantică

Semantica dă "înţeles" unui program.

- Operaţională:
  - Înţelesul programului este definit în funcţie de paşii (transformări dintr-o stare în alta) care apar în timpul execuţiei.
- Denotaţională:
  - Înţelesul programului este definit abstract ca element dintr-o structură matematică adecvată.

Tipuri de semantică

Semantica dă "înţeles" unui program.

- Operaţională:
  - Înţelesul programului este definit în funcţie de paşii (transformări dintr-o stare în alta) care apar în timpul execuţiei.
- Denotaţională:
  - Înţelesul programului este definit abstract ca element dintr-o structură matematică adecvată.
- Axiomatică:
  - Înţelesul programului este definit indirect în funcţie de axiomele şi regulile pe care le verifică.

Tipuri de semantică

Semantica dă "înţeles" unui program.

- Operaţională:
  - Înţelesul programului este definit în funcţie de paşii (transformări dintr-o stare în alta) care apar în timpul execuţiei.
- Denotaţională:
  - Înţelesul programului este definit abstract ca element dintr-o structură matematică adecvată.
- Axiomatică:
  - Înţelesul programului este definit indirect în funcţie de axiomele şi regulile pe care le verifică.
- Statică / a tipurilor
  - Reguli de bună-formare pentru programe
  - Oferă garanții privind execuția (e.g., nu se blochează)

Bazele programării funcționale / logice

Principalele paradigme de programare

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

Fundamentele paradigmelor de programare

Imperativă	Execuția unei Mașini Turing
Funcțională	Beta-reducție în Lambda Calcul
Logică	Rezoluția în logica clauzelor Horn

Programare declarativă

- Programatorul spune ce vrea să calculeze, dar nu specifică concret cum calculează.
- Este treaba interpretorului (compilator/implementare) să identifice cum să efectueze calculul respectiv.
- Tipuri de programare declarativă:
  - Programare funcţională (e.g., Haskell)
  - Programare logică (e.g., Prolog)
  - Limbaje de interogare (e.g., SQL)

Programare funcțională

Esență: funcții care relaționează intrările cu ieșirile

Caracteristici:
- funcții de ordin înalt – funcții parametrizate de funcții
- grad înalt de abstractizare (e.g., functori, monade)
- grad înalt de reutilizarea codului — polimorfism

Fundamente:
- Teoria funcțiilor recursive
- Lambda-calcul ca model de computabilitate (echivalent cu mașina Turing)

Inspirație:
- Inferența tipurilor pentru templates/generics in POO
- Model pentru programarea distribuită/bazată pe evenimente (callbacks)

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control (R. Kowalski)

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control (R. Kowalski)

- Programarea logică poate fi privită ca o deducţie controlată.

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control (R. Kowalski)

- Programarea logică poate fi privită ca o deducţie controlată.
- Un program scris într-un limbaj de programare logică este

  o listă de formule într-o logică

  ce exprimă fapte şi reguli despre o problemă.

Programare logică

- Programarea logică este o paradigmă de programare bazată pe logică formală.
- Unul din sloganurile programării logice:

  Program = Logică + Control (R. Kowalski)

- Programarea logică poate fi privită ca o deducţie controlată.
- Un program scris într-un limbaj de programare logică este

  o listă de formule într-o logică

  ce exprimă fapte şi reguli despre o problemă.
- Exemple de limbaje de programare logică:
  - Prolog
  - Answer set programming (ASP)
  - Datalog

Programare logică & Prolog

Programare logică - în mod idealist

- Un "program logic" este o colecţie de proprietăţi presupuse (sub formă de formule logice) despre lume (sau mai degrabă despre lumea programului).
- Programatorul furnizează şi o proprietate (o formula logică) care poate să fie sau nu adevărată în lumea respectivă (întrebare, query).
- Sistemul determină dacă proprietatea aflată sub semnul întrebării este o consecinţă a proprietăţilor presupuse în program.
- Programatorul nu specifică metoda prin care sistemul verifică dacă întrebarea este sau nu consecinţă a programului.

Exemplu de program logic

$$
\begin{aligned}
\texttt{oslo} &\rightarrow \texttt{windy} \\
\texttt{oslo} &\rightarrow \texttt{norway} \\
\texttt{norway} &\rightarrow \texttt{cold} \\
\texttt{cold} \wedge \texttt{windy} &\rightarrow \texttt{winterIsComing} \\
&\qquad\quad \texttt{oslo}
\end{aligned}
$$

Exemplu de program logic

$$
\begin{aligned}
\texttt{oslo} &\rightarrow \texttt{windy} \\
\texttt{oslo} &\rightarrow \texttt{norway} \\
\texttt{norway} &\rightarrow \texttt{cold} \\
\texttt{cold} \wedge \texttt{windy} &\rightarrow \texttt{winterIsComing} \\
&\rightarrow \texttt{oslo}
\end{aligned}
$$

Exemplu de întrebare

Este adevărat `winterIsComing`?

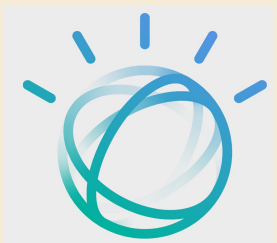
Prolog

- bazat pe logica clauzelor Horn
- semantica operaţională este bazată pe rezoluţie
- este Turing complet

Prolog

- bazat pe logica clauzelor Horn
- semantica operaţională este bazată pe rezoluţie
- este Turing complet

Limbajul Prolog este folosit pentru programarea sistemului IBM Watson!



Puteţi citi mai multe detalii aici.

Exemplul de mai sus în SWI-Prolog

Program:

```
windy :- oslo.
norway :- oslo.
cold :- norway.
winterIsComing :- windy, cold.
oslo.
```

Intrebare:

```
?- winterIsComing.
true
```

http://swish.swi-prolog.org/