Curs 1

2021-2022

Fundamentele Limbajelor de Programare

Cuprins

Organizare

Instructori

Curs

- **Ioana Leuștean (seriile 23, 25)**
- **Denisa Diaconescu (seria 24)**

Laborator

Seria 23
- **Ana Iova (Țurlea) (231)**
- **Horatiu Cheval (232)**
- **Bogdan Macovei (233,234)**

Seria 24
- **Natalia Ozunu (241)**
- **Bogdan Macovei (242,243,244)**

Seria 25
- **Ana Iova (Țurlea) (251)**
- **Bogdan Macovei (252)**

Resurse

- Seriile 23, 25
  - Moodle
  - Pagina externa
- Seria 24
  - Moodle
  - Pagina externa
- Suporturile de curs si laborator/seminar, resurse electronice
- Stiri legate de curs vor fi postate pe Moodle si pe paginile externe

Prezenta

Prezenta la curs sau la laboratoare/seminarii nu este obligatorie, dar extrem de incurajata.

Notare

Notare

- Nota finala: 1 punct (oficiu) + examen

Notare

- Nota finala: 1 punct (oficiu) + examen
- Conditie minima pentru promovare: nota finala > 4.99

Notare

- Nota finala: 1 punct (oficiu) + examen
- Conditie minima pentru promovare: nota finala > 4.99
- Puncte bonus
  - La sugestia profesorului coordonator al laboratorului/seminarului, se poate nota activitatea în plus fata de cerintele obisnuite
  - Maxim 1 punct

Examen

- In sesiunea
- Durata 2 ore
- Cu materialele ajutatoare
- Mai multe detalii vor fi oferite pana la jumatatea semestrului

Curs/laborator

- Curs

  Bazele programării logice
  - Logica clauzelor Horn, Unificare, Rezolutie

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

  Bazele programării funcționale
  - Lambda Calcul, Codificări Church, corespondenta Curry-Howard
  - Lambda Calcul cu tipuri de date

Curs/laborator

- Curs

  Bazele programării logice
  - Logica clauzelor Horn, Unificare, Rezolutie

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

  Bazele programării funcționale
  - Lambda Calcul, Codificări Church, corespondenta Curry-Howard
  - Lambda Calcul cu tipuri de date

- Laborator/Seminar:

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

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

  Exercitii suport pentru curs

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

Bazele programării funcționale / logice

Principalele paradigme de programare

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

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

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 functională (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 deductie 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 deductie controlată.
- Un program scris intr-un limbaj de programare logică este

  o listă de formule intr-o logică

  ce exprimă fapte și 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 deductie controlată.
- Un program scris intr-un limbaj de programare logică este

  o listă de formule intr-o logică

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

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)

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)

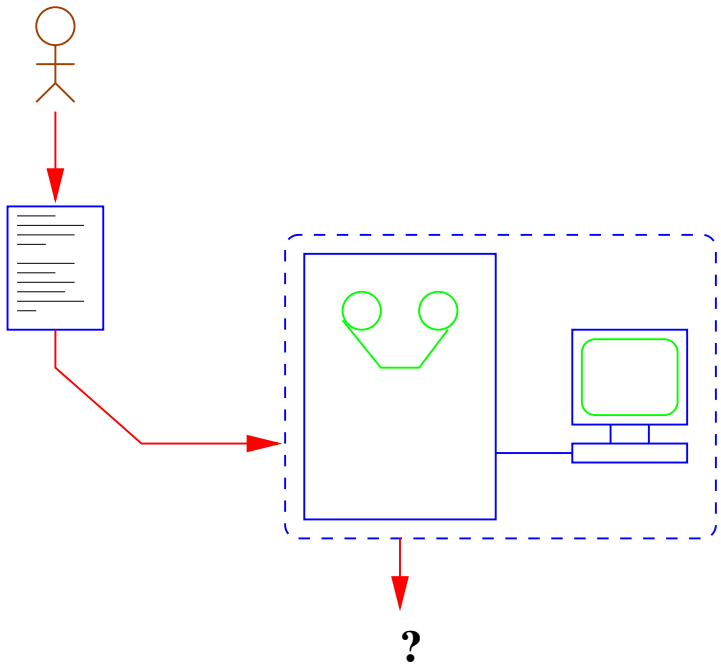
Semantica? Ce înseamnă / care e comportamentul unei instrucțiuni?

- 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 ușor să stabilim că 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);
}
```

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 "înțeles" unui program.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.
- Denotațională:
  - Înțelesul programului este definit abstract ca element dintr-o structură matematică adecvată.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.
- Denotațională:
  - Înțelesul programului este definit abstract ca element dintr-o structură matematică adecvată.
- Axiomatică:
  - Înțelesul programului este definit indirect în funcție de axiomele și regulile pe care le verifică.

Tipuri de semantică

Semantica dă "înțeles" unui program.

- Operațională:
  - Înțelesul programului este definit în funcție de pașii (transformări dintr-o stare în alta) care apar în timpul execuției.
- Denotațională:
  - Înțelesul programului este definit abstract ca element dintr-o structură matematică adecvată.
- Axiomatică:
  - Înțelesul programului este definit indirect în funcție de axiomele și regulile pe care le verifică.
- Statică / a tipurilor
  - Reguli de bună-formare pentru programe
  - Oferă garanții privind execuția (e.g., nu se blochează)

Programare logică & Prolog

Programare logică - în mod idealist

- Un "program logic" este o colecție de proprietăți presupuse (sub formă de formule logice) despre lume (sau mai degrabă despre lumea programului).
- Programatorul furnizează și 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 consecință a proprietăților presupuse în program.
- Programatorul nu specifică metoda prin care sistemul verifică dacă întrebarea este sau nu consecință a programului.

Exemplu de program logic

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

Exemplu de program logic

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

Exemplu de întrebare

Este adevărat `winterIsComing`?

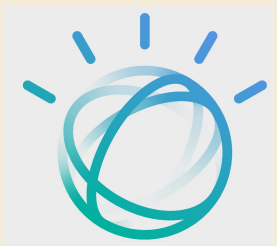
Prolog

- bazat pe logica clauzelor Horn
- semantica operațională este bazată pe rezoluție
- este Turing complet

Prolog

- bazat pe logica clauzelor Horn
- semantica operațională este bazată pe rezoluție
- este Turing complet

Limbajul Prolog este folosit pentru programarea sistemului IBM Watson!



Puteți 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/

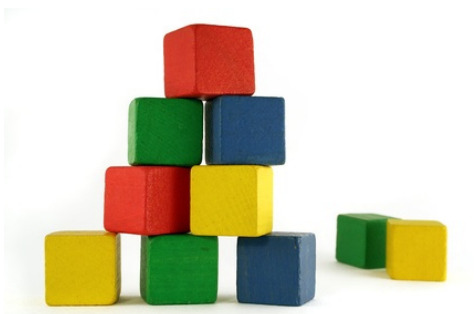
Quiz time!

https://www.questionpro.com/t/AT4NiZrHFn

Sintaxă: constante, variabile, termeni compuși

- Atomi: `brian, 'Brian Griffin', brian_griffin`
- Numere: `23, 23.03,-1`

  Atomii și numerele sunt constante.
- Variabile: `X, Griffin, _family`
- Termeni compuși: `father(peter, stewie_griffin)`,
  `and(son(stewie,peter), daughter(meg,peter))`
  - forma generală: atom(termen,. . ., termen)
  - atom-ul care denumește termenul se numește functor
  - numărul de argumente se numește aritate



Un mic exercițiu sintactic

Care din următoarele șiruri de caractere sunt constante și care sunt variabile în Prolog?

- vINCENT
- Footmassage
- variable23
- Variable2000
- big_kahuna_burger
- 'big kahuna burger'
- big kahuna burger
- 'Jules'
- _Jules
- '_Jules'

Un mic exercițiu sintactic

Care din următoarele șiruri de caractere sunt constante și care sunt variabile în Prolog?

- vINCENT – constantă
- Footmassage – variabilă
- variable23 – constantă
- Variable2000 – variabilă
- big_kahuna_burger – constantă
- 'big kahuna burger' – constantă
- big kahuna burger – nici una, nici alta
- 'Jules' – constantă
- _Jules – variabilă
- '_Jules' – constantă

Program în Prolog = bază de cunoștințe

Example

Un program în Prolog:

```
father(peter,meg).
father(peter,stewie).

mother(lois,meg).
mother(lois,stewie).

griffin(peter).
griffin(lois).

griffin(X) :- father(Y,X),  griffin(Y).
```

Un program în Prolog este o bază de cunoștințe (Knowledge Base).

Program în Prolog = mulțime de predicate

Practic, gândim un program în Prolog ca o mulțime de predicate cu ajutorul cărora descriem *lumea* (*universul*) programului respectiv.

Example

```
father(peter,meg).
father(peter,stewie).

mother(lois,meg).
mother(lois,stewie).

griffin(peter).
griffin(lois).

griffin(X) :- father(Y,X),  griffin(Y).
```

Predicate:
father/2
mother/2
griffin/1

Un program în Prolog

Program

- Un program în Prolog este format din reguli de forma

```
Head :- Body.
```

- Head este un predicat, iar Body este o secvență de predicate separate prin virgulă.
- Regulile fără `Body` se numesc fapte.

Program

- Un program în Prolog este format din reguli de forma

```
Head :- Body.
```

- Head este un predicat, iar Body este o secvență de predicate separate prin virgulă.
- Regulile fără `Body` se numesc fapte.

Example

- Exemplu de regulă: `griffin(X) :- father(Y,X), griffin(Y).`
- Exemplu de fapt: `father(peter,meg).`

Interpretarea din punctul de vedere al logicii

- operatorul `:-` este implicația logică ←

Example

```
comedy(X) :- griffin(X)
```

dacă `griffin(X)` este adevărat, atunci `comedy(X)` este adevărat.

Interpretarea din punctul de vedere al logicii

- operatorul `:-` este implicația logică $\leftarrow$

Example

```
comedy(X) :- griffin(X)
```

dacă `griffin(X)` este adevărat, atunci `comedy(X)` este adevărat.

- virgula `,` este conjuncția $\wedge$

Example

```
griffin(X) :- father(Y,X), griffin(Y).
```

dacă `father(Y,X)` și `griffin(Y)` sunt adevărate,
atunci `griffin(X)` este adevărat.

Interpretarea din punctul de vedere al logicii

- mai multe reguli cu același `Head` definesc același predicat, între defiții fiind un sau logic.

Example

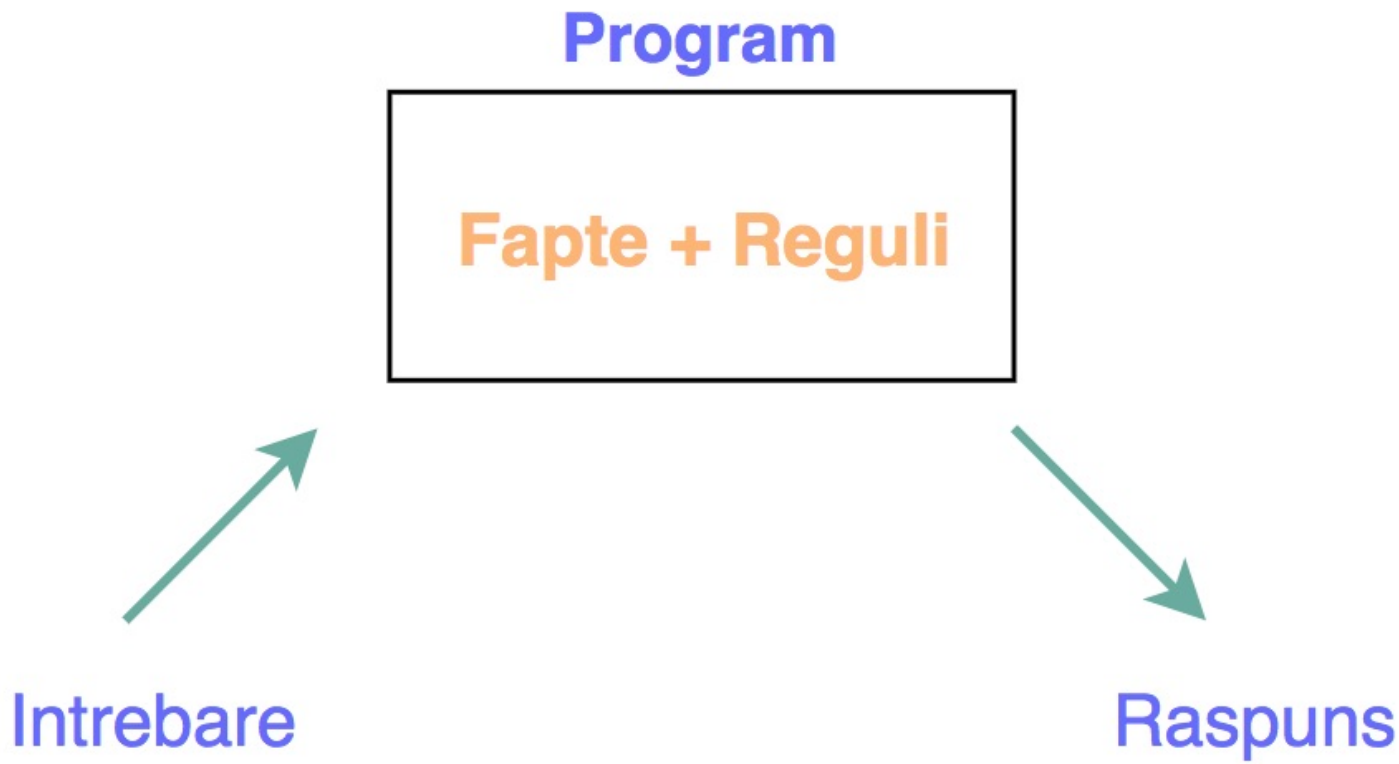
```
comedy(X) :- family_guy(X).
comedy(X) :- south_park(X).
comedy(X) :- disenchantment(X).
```

dacă
`family_guy(X)` este adevărat sau `south_park(X)` este adevărat sau `disenchantment(X)` este adevărat,
atunci
`comedy(X)` este adevărat.

Un program în Prolog

Cum folosim un program în Prolog?

Întrebări în Prolog



Întrebări și ținte în Prolog

- Prolog poate răspunde la întrebări legate de consecințele relațiilor descrise într-un program în Prolog.
- Întrebările sunt de forma:

$$\texttt{?- predicat}_1\texttt{(...),...,predicat}_n\texttt{(...).}$$

- Prolog verifică dacă întrebarea este o consecință a relațiilor definite în program.
- Dacă este cazul, Prolog caută valori pentru variabilele care apar în întrebare astfel încât întrebarea să fie o consecință a relațiilor din program.
- Un predicat care este analizat pentru a se răspunde la o întrebare se numește țintă (goal).

Întrebări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- `false` – în cazul în care întrebarea nu este o consecință a programului.
- `true` sau valori pentru variabilele din întrebare în cazul în care întrebarea este o consecință a programului.

Întrebări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- `false` – în cazul în care întrebarea nu este o consecință a programului.
- `true` sau valori pentru variabilele din întrebare în cazul în care întrebarea este o consecință a programului.

Example

```
?- griffin(meg)
true
?- griffin(glenn)
false
```

```
?- griffin(X)
X = petr ;
X = lois ;
X = meg ;
X = stewie ;
false
```

Pe săptămâna viitoare!