Organizare

Organizare

- Instructori
  - **Ioana Leuștean**
  - **Bogdan Macovei**
- Resurse
  - Suporturile de curs și laborator/seminar, resurse electronice
  - Stiri legate de curs vor fi postate pe grupul dedicat de Teams.

Evaluare

- **Parțial**: în săptămâna 14-17 aprilie (pe laptopul propriu)
  1. Programare Prolog
     - Durata: 1 oră
     - Punctaj maxim: 30 puncte
     - Punctaj promovare: 10 puncte
     - Nu poate fi dat in sesiune
- **Examen**: în sesiune
  2. Programare Haskell
     - Durata: 1 oră
     - Punctaj maxim Haskell: 30 puncte
     - Punctaj promovare Haskell: 15 puncte
  3. Teorie
     - Durata: 1 oră
     - Punctaj maxim Teorie: 30 puncte
     - Punctaj promovare Teorie: 15 puncte

Materiale pentru examen. Condiții de promovare.

- La evaluări studenții nu au acces la materialele de curs, seminar, laborator și nici la alte materiale disponibile online. Studenții vor avea acces la un material redactat de instructori special pentru evaluări.
- Conditia de promovare a examenului: obținerea celor trei punctaje de promovare.
- Nota finală se obține din suma celor trei punctaje - Programare Prolog, Programare Haskell, Teorie - la care se adauga punctajul suplimentar acordat pentru activitatea la curs/seminar/laborator si 10 puncte din oficiu. Punctele din oficiu nu se iau în calcul la punctajul pentru promovare.

Absențe, restanțe, măriri

- Un student va fi absent la finalul sesiunii numai dacă a fost absent atât la Parțial, cât și la Examenul din sesiune.
- Studenții care **nu îndeplinesc condiția de promovare** au o notă ≤ 4, dar punctajele de promovare parțiale se păstrează în sesiunile de restanțe.
- Punctajele de promovare obținute la fiecare dintre cele trei părți - Programare Prolog, Programare Haskell, Teorie - se pot mări numai în sesiunea de măriri (septembrie).

Toate regulile privind evaluarea și promovarea sunt valabile numai pentru anul universitar 2024-2025.

Curs/laborator

- Programare logică
  - Limbajul Prolog
  - Logica propozițională, logica de ordinul I, logica clauzelor Horn
  - Unificare
  - Rezolutie
- Programare funcțională
  - Limbajul Haskell
    - Funcții și recursie
    - Definirea listelor și procesarea lor
    - Tipuri de date algebrice. Clase de tipuri
    - Operațiuni de intrare-ieșire
    - Introducere în utilizarea monadelor.
  - Introducere în Lambda Calcul
- Introducere în semantica limbajelor de programare

Bibliografie

- B.C. Pierce, **Types and programming languages**. MIT Press.2002
- C. Allen, J. Moronuki, Haskell Programming From First Principles, Gumrod 2015.
- H. Barendregt, E. Barendsen, **Introduction to Lambda Calculus**, 2000.
- J. Lloyd. **Foundations of Logic Programming**, second edition. Springer, 1987.
- M. Huth, M. Ryan, **Logic in Computer Science (Modelling and Reasoning about Systems)**, Cambridge University Press, 2004.
- P. Blackburn, J. Bos, and K. Striegnitz, **Learn Prolog Now!** (Texts in Computing, Vol. 7),College Publications, 2006
- `https://lpn.swi-prolog.org/lpnpage.php?pageid=online`
- `https://www.haskell.org/documentation/`

Principalele paradigme de programare

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

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

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 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.

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.

Vom folosi limbajele Prolog și Haskell pentru
a defini un mini limbaj și semantica lui!

Proiecte

Proiecte

(p1) [**Prolog**] Implementați rezoluția din calculul propozițional clasic folosind algoritmul Davis-Putnam. Variantă mai complexă: implementați un SMT solver.
Referințe:
[1] J. Howe, A. King, A pearl on SAT and SMT solving in Prolog, TCS 435, 2010

(p2) [**Prolog**] Implementati un demonstrator pentru logica de ordinul I, folosind Teorema lui Herbrand si rezolutia pe clauze închise (ground resolution).
Referințe:
[1] M.H. Van Emden, R.W. Kowalski, The Semantics of Predicate Logic as a Programing Language, JACM 733-742, 1976

Proiecte

(p3) [**Prolog**] Implementați în Prolog deducția naturală.
Referințe:
[1] `https://fse.studenttheses.ub.rug.nl/25522/1/bAI_2021_LijnzaadFJA.pdf`

(p4) [**Prolog**] Implementați un chatbot care să răspundă la întrebări legate de facultate. Alternativ puteti propune o temă care vă interesează, dar diferită de cea orginală.
Referințe:
[1] J. Weizenbaum, ELIZA A Computer Program For the Study of Natural Language Communication Between Man And Machine, Comm. ACM 9,1966,
`https://dl.acm.org/doi/pdf/10.1145/365153.365168`
[2] `https://like-a-boss.net/2018/08/24/eliza-a-tutorial-reconstruction-in-prolog.html`

Proiecte

(p5) [**Prolog/Python**] Scrieți un program Prolog pentru a genera orarul pentru CTI (varianta complexă: pentru FMI).
Referințe:
[1] SWI-Prolog Python interface
[2] R. Fahrion, G. Dollansky, Construction of university faculty timetables using logic programming techniques, Discrete Applied Mathematics 25, 1992 `https://www.sciencedirect.com/science/article/pii/0166218X92902467`
[3] J. Siang Tan, S. Leng Goh, G. Kendall, N. R. Sabar, A survey of the state-of-the-art of optimisation methodologies in school timetabling problems, Expert Systems With Applications 165, 2021, `https://www.graham-kendall.com/papers/tgks2021.pdf` (conține multe referințe)

Proiecte

(p6) [**Prolog/Haskell**] Implementați un verificator de demonstratii (proof checker) pentru demonstrații folosind sistmul de deducție Hilbert in calculul propoziținal și in logica de ordinul I. Verificatorul va primi o listă de formule și va verfica dacă lista este o demonstrație formală în sistemul menționat.

(p7) [**Prolog/Haskell**] Definiți un mini-limbaj de asamblare și scrieți un compilator pentru limbajul IMP definit la curs.
Referințe:
[1] J. Cohen and T. Hickey, Parsing and Compiling using Prolog, ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 9, No. 2, pp.125-163, 1987
`https://dl.acm.org/doi/pdf/10.1145/22719.22946`
[2] D.H.D. Warren, Logic programming and compiler writing, Software Practice and Experience, Vol. 10, pp. 97-125, 1980
[3] J.P. Bowen, From Programs to Object Code using Logic and Logic Programming, Code Generation — Concepts, Tools, Techniques, pp. 173–192, 1994

Proiecte

(p8) [**Haskell**] Implementați algoritmul de unificare și rezolutia SLD.

(p9) [**Prolog/Haskell**] Extindenți limbajul IMP cu diferite elemente (intrări și ieșiri, vectori, operatții și instrucțiuni, funcții, clase și obiecte) sau definiți un limbaj nou (sintaxa și semantică executabilă).

(p10) [**Prolog/Haskell**] Propunere proprie (discutată în prealabil cu instructorii)