Curs 1

2016-2017

Programare Logică

Cuprins

Organizare

Instructori

- **Denisa Diaconescu**
- **Carmen Chirița**
- **Claudia Mureșan**
- **Alexandra Otiman**
- **Andrei Sipoș**
- **Ana Țurlea**

Suport curs

- Moodle
- https://sites.google.com/view/pl2017/home

  sau

  https://goo.gl/tQpnUi

Bibliografie

- J. Goguen, **Theorem Proving and Algebra**, manuscris.
- F. Baader, T. Nipkow, **Terms Rewriting and All That**, Cambridge University Press, 1998.
- F.L. Țiplea, **Fundamentele algebrice ale informaticii**, (II40405, biblioteca FMI).
- V.E. Căzănescu, **Note de curs**.

Notare

Notare

- **Laborator: 30 puncte**
- **Examen: 70 puncte**

- Condiție minină pentru promovare: cel puțin 50% la fiecare probă
  - laborator: min. 15 puncte și
  - examen: min. 35 puncte

Laborator: 30 puncte

Parțial:

- Are loc în Săptămâna 9 (24 – 28 aprilie)
- Se va susține în cadrul cursului
- Prezența la parțial este obligatorie!!
- Nu se poate reface (doar la restanță)
- Timp de lucru: 45 min

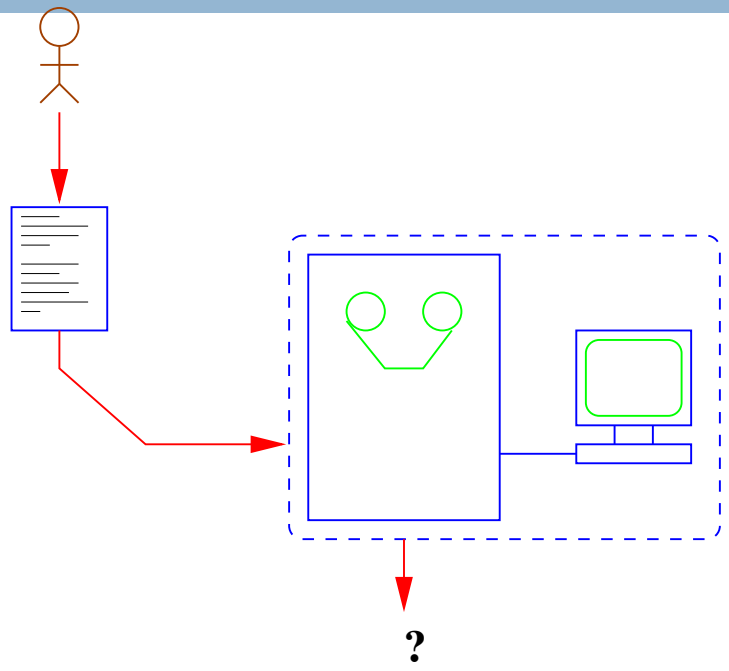
Examen: 70 puncte

- Subiecte de teorie și exerciții.
- Timp de lucru: 2 ore
- În Săptămâna 14 veți primi o foaie cu teorie pe care o veți primi si la examen!
- Subiectele de teorie constau în demonstrarea unor rezultate din curs (demonstrate la curs sau lăsate ca temă).
- Subiectele de exerciții vor fi în stilul celor rezolvate la seminar (în Laboratoarele 9-14).
- La examen, trebuie să obțineți min. 35 puncte.

Privire de ansamblu

Curs

Problema corectitudinii programelor



- Pentru metodele convenționale de programare (imperative), nu este ușor să vedem că un program este corect sau să înțelegem ce înseamnă că este corect (în raport cu ce?!).
- 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".

Un program imperativ simplu

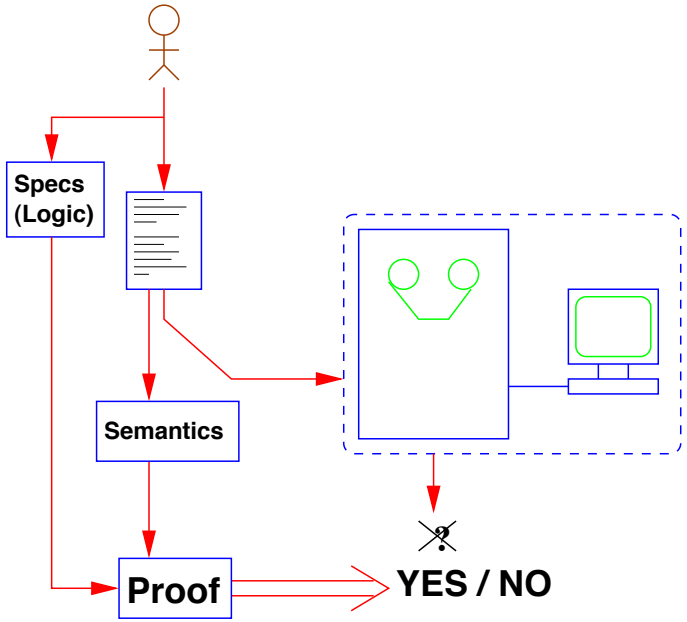
```
#include <iostream>
using namespace std;
main()
{
  int square;
  for(int i = 1; i <= 5; ++i)
  {
    square = i * i;
    cout <<  square << endl;
  }
}
```

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

Logica

- Un mijloc de a clarifica/modela procesul de a "raționa".
- Care logică?
  - propozițională
  - de ordinul I
  - de ordin înalt
  - logici modale
  - λ-calcul
  - logici cu mai multe valori
  - ...

Folosind logica



Logica ne permite să reprezentăm/modelăm probleme.

Pentru a scrie specificații și a raționa despre corectitudinea programelor:

- Limbaje de specificații (modelarea problemelor)
- Semantica programelor (operațională, denotațională, ...)
- Demonstrații (verificarea programelor, ...)

Semantica unui program

- Semantica dă un "înțeles" (obiect matematic) unui program.
- Semantica trebuie:
  - să poată verifica că un program satisface condițiile cerute.
  - să poată demonstra că două programe au aceeași semantica.
  - ...

Tipuri de Semantică

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

Programare Logică

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

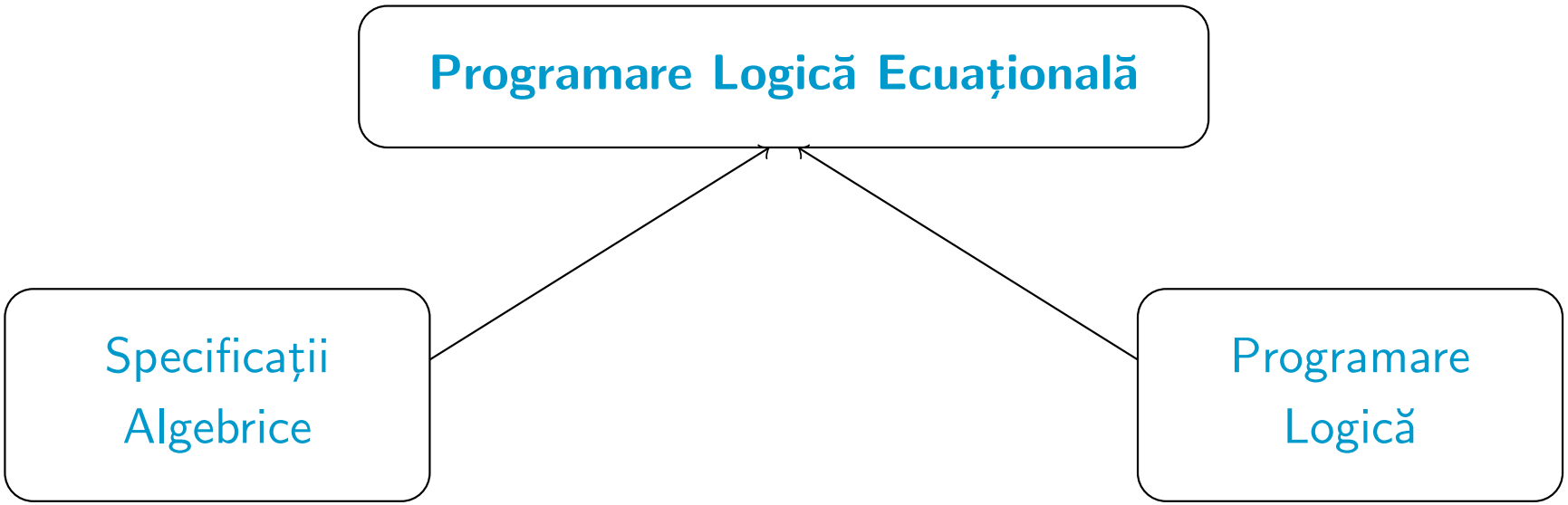
  Program = Logică + Control *(R. Kowalski)*

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

  o listă de formule într-o logică

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

Programare Logică Ecuațională



- Programarea logică ecuațională unește
  - programarea logică bazată pe clauze Horn *(Prolog)*
  - programarea ecuațională bazată pe logica ecuațională
- Câștig: Putem folosi egalitatea!

Specificații algebrice

- Tipurile abstracte de date (ADTs) sunt obiecte matematice pentru tipuri de date
  - un tip de date este definit prin comportamentul său (semantica) din punctul de vedere al utilizatorului
  - comportamentul este specificat în funcție de valorile posibile, operațiile posibile pe tipul de date etc.
- ADT sunt des întalnite în dezvoltarea de aplicații/limbaje
- Tipuri abstracte de date vs. structuri de date
  - structurile de date sunt o reprezentare concretă a datelor și sunt punctul de vedere al unui limbaj, nu al utilizatorului.
- Specificațiile algebrice oferă un cadru matematic pentru a defini ADTs

Ce veți vedea la curs

1. Programare logică clasică
2. Specificații algebrice
3. Programare logică ecuațională
4. Sisteme de rescriere

Laborator

Ce veți vedea la laborator

Prolog:

- cel mai cunoscut limbaj de programare logică
- bazat pe un fragment al logicii clasice (logica Horn)
- semantica operațională este bazată pe rezoluție
- vom folosi implementarea SWI-Prolog
- vom folosi varianta online SWISH a SWI-Prolog
  - `http://swish.swi-prolog.org/`

Ce veți vedea la laborator

Maude:

- un limbaj de specificații executabil
- un fragment este bazat pe logica ecuațională
- semantica operațională este bazată pe rescriere
- `http://maude.cs.uiuc.edu/`
- Maude REPL: `http://maude.cvlad.info/`

Ce veți vedea la laborator

Exerciții suport pentru curs.

Planificare laboratoare

- Săptămânile 1 - 4: Limbajul Prolog
- Săptămânile 5 - 8: Limbajul Maude
- Săptămânile 9 - 14: Seminarii - exerciții suport pentru curs


LET'S GET STARTED

Programare logică clasică

Programare declarativă

- Programare procedurală
  - Un program descrie explicit fiecare pas al unui calcul.
  - **Exemple:**
    - programare imperativă (C)
    - programare orientată-obiect (Java)
- 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.
  - Programarea logică este un tip de programare declarativă!
  - **Exemple:**
    - programare logică (Prolog)
    - programare funcțională (Haskell)

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.

Programare logică - în mod idealist

Aspecte declarative ale programării logice:

- Programatorul nu specifică metoda prin care sistemul verifică dacă întrebarea este sau nu consecință a programului.
- Faptul că întrebarea chiar este sau nu consecință este independent de metoda aleasă de sistem.

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`?

Putem să testăm în SWI-Prolog

Program:

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

Intrebare:

```
?- winterIsComing.
true
```

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

Puncte-cheie

- De ce acesta este răspunsul corect?
- Cum calculează Prolog răspunsul?
- Găsește mereu Prolog răspunsul corect?

1. Cazul logicii propoziționale
2. Cazul calculul cu predicate

Cazul logicii propoziționale

Logica propoziţională (recap.)

- Formule:

$$form \;::=\; atom \mid \neg form \mid form \wedge form \mid form \vee form \mid form \rightarrow form$$

- Un exemplu de formulă:

$$(\texttt{stark} \wedge \neg\, \texttt{dead}) \rightarrow (\texttt{sansa} \vee \texttt{arya} \vee \texttt{bran})$$

- Formulele pe care le vom folosi în "programele logice" au o structură foarte simpla.
  Vom vedea mai tarziu că acest lucru nu este întâmplător!

Interpretări

- O interpretare este o funcție care dă valori de adevăr (**true** și **false**) atomilor.
- De exemplu, dacă atomii sunt `poor` și `happy` atunci

$$\{\ \texttt{poor} \mapsto \textbf{false}, \texttt{happy} \mapsto \textbf{true}\ \}$$

  este o interpretare.
- O formulă F este adevărată (**true**) într-o interpretare $\mathcal{I}$, notând

$$\mathcal{I} \models F,$$

  dacă valoarea de adevăr a formulei obținută folosind tabelele de adevăr și interpretarea $\mathcal{I}$ este **true**.
- De exemplu,

$$\{\ \texttt{poor} \mapsto \textbf{false}, \texttt{happy} \mapsto \textbf{true}\ \} \models \texttt{poor} \rightarrow \texttt{happy}$$

Consecință logică

- O formulă G este o consecință logică a unor formule $F_1, F_2, \ldots, F_n$,

$$F_1, \ldots, F_n \models G,$$

dacă pentru orice interpretare $\mathcal{I}$

dacă $\mathcal{I} \models F_1$ și ... și $\mathcal{I} \models F_n$, atunci $\mathcal{I} \models G$.

- Atenție! Simbolul $\models$ este folosit în două moduri diferite:

$$\begin{array}{rcl} \mathcal{I} & \models & G \\ F_1, \ldots, F_n & \models & G \end{array}$$

În primul caz, partea stângă este o interpretare, iar în al doilea este o mulțime de formule.

Exemple

- Formulă care este o consecință logică:

$$\texttt{poor} \to \texttt{happy}, \neg\, \texttt{happy} \models \neg\texttt{poor}$$

Dacă ne uităm la toate interpretările (sunt 4!), observăm că de fiecare dată atât formulele din stanga, cât și formula din dreapta sunt adevărate.

- Formulă care nu este o consecință logică:

$$\texttt{poor} \to \texttt{happy}, \neg\texttt{poor} \not\models \neg\, \texttt{happy}$$

Interpretarea $\{\texttt{poor} \mapsto \textbf{false}, \texttt{happy} \mapsto \textbf{true}\}$ face ambele formule din stânga adevărate, dar $\neg\, \texttt{happy}$ este falsă.

Ideea unui "program logic" (propozițional)

- Un "program logic" este o listă de formule (propoziționale)

$$F_1, F_2, \ldots, F_n.$$

- O țintă (*goal*) este o formulă

$$G.$$

- Sistemul trebuie să determine dacă

$$F_1, \ldots, F_n \models G.$$

  Dacă da, sistemul returnează `"yes"`/`"true"`. Altfel, returnează `"no"`/`"false"`.

Verificarea problemei consecinței logice

- În principiu, sistemul poate verifica problema consecinței logice construind un tabel de adevăr, cu câte o linie pentru fiecare interpretare posibilă.
- În cazul în care programul și ținta conțin n atomi diferiți, tabelul de adevăr rezultat o să aibă 2^n rânduri.
- Această metodă este atât de costisitoare computațional, încât este irealizabilă. (Timp exponențial)
- Problemă deschisă de un milion de dolari:

  Este posibil să găsim o metodă mai bună pentru a decide problema consecinței logice în cazul propozițional care să funcționeze în timp polinomial față de intrarea programului și a țintei?

  Echivalent, este adevărată $P = NP$?
  (Institutul de Matematica Clay – Millennium Prize Problems)

Cum salvăm situția?

1. Restricționarea formulelor din "programele logice" (clauze definite)
2. Folosirea metodelor sintactice pentru a stabili problema consecinței logice (*proof search*)

- Această metodologie este eficientă și flexibilă.
- În următoarele cursuri o vom extinde la calculul cu predicate (și nu numai!).

Clauze propoziționale definite

- O clauză definită este o formulă care poate avea una din formele:

  1. q (clauza unitate) (un fapt în Prolog `q.`)
  2. $p_1 \wedge \ldots \wedge p_k \rightarrow q$ (o regulă în Prolog `q :- p1,...,pn`)

  unde $q, p_1, \ldots, p_n$ sunt atomi.

- Un "program logic" este o listă $F_1, \ldots, F_n$ de clauze definite.

- O țintă (*goal*) este o listă $g_1, \ldots, g_m$ de atomi.

- Sarcina sistemului este să stabilească:

$$F_1, \ldots, F_n \models g_1 \wedge \ldots \wedge g_m.$$

Pe săptămâna viitoare!