Fundamentele limbajelor de programare

C01

Denisa Diaconescu
Traian Șerbănuță

Departamentul de Informatică, FMI, UB

Organizare

Instructori

- Curs
  - Seria 23: Traian Șerbănuță
  - Seria 24: Denisa Diaconescu
  - Seria 25: Traian Șerbănuță
- Laborator
  - 231: Horațiu Cheval
  - 232: Horațiu Cheval/Bogdan Macovei
  - 233: Andrei Văcaru
  - 234: Horațiu Cheval/Bogdan Macovei
  - 241: Natalia Ozunu
  - 242: Bogdan Macovei
  - 243: Bogdan Macovei
  - 244: Bogdan Macovei
  - 251: Mihai Calancea
  - 252: Andrei Burdușa

Resurse

- Moodle
- Teams
  https://tinyurl.com/FLP2023-Teams
- Suporturile de curs si laborator
  https://tinyurl.com/FLP2023-Materials

Prezență

Prezența la curs sau la laboratoare nu este obligatorie, dar extrem de încurajată.

Evaluare

Notare

- Nota finală: 1 (oficiu) + nota laborator + parțial + examen
- Restanță: 1 (oficiu) + examen
  (nota de la laborator si parțialul nu se iau în calcul la restanță)

Condiție de promovabilitate

- cel puțin 5 > 4.99

Nota laborator

- valorează 2 puncte din nota finală
- se notează activitatea din cadrul laboratorului

Examen parțial

- valorează 3 puncte din nota finală
- durata 30 min
- în săptămâna 7, în cadrul cursului
- nu este obligatoriu și nu se poate reface
- întrebări grilă asemănatoare cu cele din quiz-urile de la curs
- materiale ajutătoare: suporturile de curs și de laborator

Examen final

- valorează 4 puncte din nota finală
- durata 1 oră
- în sesiune, fizic
- acoperă toată materia
- exerciții asemănătoare cu exemplele de la curs (nu grile)
- materiale ajutătoare: suporturile de curs și de laborator

Imagine de ansamblu asupra materiei

Curs

- Partea I
  - Lambda calcul
  - Deducție naturală
  - Corespondența Curry-Howard
- Partea II
  - Puncte fixe/recursivitate
  - Semantica limbajelor de programare
  - Elemente de programare logică*

Laborator

- Limbajul suport: Haskell
- Parsere
- Type-checking
- Implementarea unor semantici de limbaje

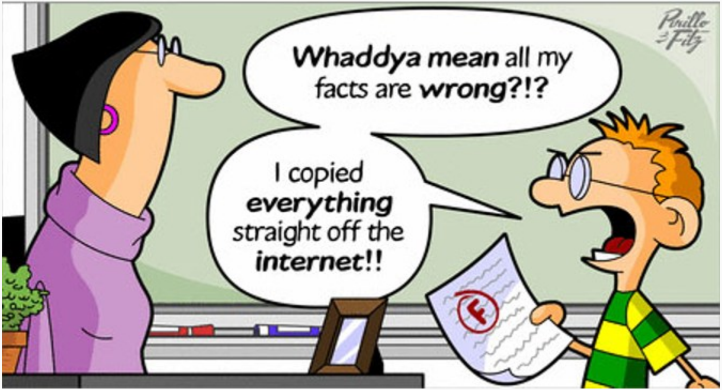
Bibliografie

- H. Barendregt, E. Barendsen, **Introduction to Lambda Calculus**, 2000.
- R. Nederpelt, H. Geuvers , **Type Theory and Formal Proof**. Cambridge University Press, 2014.
- B.C. Pierce, **Types and programming languages**. MIT Press, 2002
- P. Selinger,**Lecture Notes on the Lambda Calculus**. Dep. of Mathematics and Statistics, Dalhousie University, Canada.
- 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.
- J. Lloyd. **Foundations of Logic Programming**, second edition. Springer, 1987.

La acest curs vom folosi destul de mult literele grecești

Αα **ALPHA** [a] *ἄλφα*	Ββ **BETA** [b] *βῆτα*	Γγ **GAMMA** [g] *γάμμα*	Δδ **DELTA** [d] *δέλτα*	Εε **EPSILON** [e] *ἒ ψιλόν*	Ζζ **ZETA** [dz] *ζῆτα*
Ηη **ETA** [ɛː] *ἦτα*	Θθ **THETA** [tʰ] *θῆτα*	Ιι **IOTA** [i] *ἰῶτα*	Κκ **KAPPA** [k] *κάππα*	Λλ **LAMBDA** [l] *λάμβδα*	Μμ **MU** [m] *μῦ*
Νν **NU** [n] *νῦ*	Ξξ **XI** [ks] *ξεῖ*	Οο **OMICRON** [o] *ὂ μικρόν*	Ππ **PI** [p] *πεῖ*	Ρρ **RHO** [r] *ῥῶ*	Σσς **SIGMA** [s] *σῖγμα*
Ττ **TAU** [t] *ταῦ*	Υυ **UPSILON** [ʉ] *ὖ ψιλόν*	Φφ **PHI** [pʰ] *φεῖ*	Χχ **CHI** [kʰ] *χεῖ*	Ψψ **PSI** [ps] *ψεῖ*	Ωω **OMEGA** [ɔː] *ὦ μέγα*

Nu trișați, cereți-ne ajutorul!



Ce și de ce lambda calcul?

Ce este o funcție în matematică?

- In matematica modernă, avem "funcții prin grafice":
  - orice funcție f are un domeniu X și un codomeniu Y fixate, și
  - orice funcție $f : X \to Y$ este o mulțime de perechi $f \subseteq X \times Y$ a.î. pentru orice $x \in X$, există exact un $y \in Y$ astfel încât $(x, y) \in f$.
- Acesta este un punct de vedere *extensional*, singurul lucru pe care îl putem observa despre funcție este cum duce intrările în ieșiri.
- Două funcții $f, g : X \to Y$ sunt considerate ca fiind *extensional* egale dacă pentru aceeași intrare obțin aceeași ieșire,

$$f(x) = g(x), \text{ pentru orice } x \in X.$$

Ce este o funcție în matematică?

- Înainte de secolul 20, funcțiile erau privite ca "reguli/formule".
- A defini o funcție înseamnă să dai o regulă/formulă pentru a o calcula. De exemplu,
$$f(x) = x^2 - 1.$$
- Doua funcții sunt *intensional* egale dacă sunt definite de aceeași formulă. De exemplu, este f de mai sus intensional egala cu g de mai jos?
$$g(x) = (x - 1)(x + 1).$$
- Daca privim o funcție ca o formulă, nu este mereu necesar să știm domeniul și codomeniul ei. De exemplu, funcția identitate
$$h(x) = x$$
poate fi privită ca o funcție $h : X \to X$, pentru orice mulțime X.

Extensional vs. intensional

- Paradigma "funcții prin grafice" este foarte elegantă și definește o clasă mai largă de funcții, deoarece cuprinde și funcții care nu pot fi definite prin formule.
- Paradigma "funcții ca formule" este utilă de multe ori în informatică. De exemplu, putem privi un program ca o funcție de la intrări la ieșiri. De cele mai multe ori, nu ne interesează doar cum sunt duse intrările în ieșiri, ci și cum o putem implementa, cum a fost calculată ieșirea, diverse informații suplimentare etc.
    - Cât a durat să o calculăm?
    - Câtă memorie a folosit?
    - Cu cine a comunicat?

O paranteză: expresii aritmetice

- Expresiile aritmetice sunt construite din
  - variabile ($x, y, z, \ldots$)
  - numere ($1, 2, 3, \ldots$)
  - operatori ("$+$", "$-$", "$\times$" etc)

- Gândim o expresie de forma $x + y$ ca rezultatul adunării lui x cu y, nu ca instrucțiunea/declarația de a aduna x cu y.

- Expresiile aritmetice pot fi combinate, fără a menționa în mod explicit rezultatele intermediare. De exemplu, scriem

$$A = (x + y) \times z^2$$

în loc de

$$\text{fie } w = x + y, \text{ apoi fie } u = z^2, \text{ apoi fie } A = w \times u.$$

Lambda calcul

- Lambda calculul este o teorie a funcțiilor ca formule.
- Este un sistem care permite manipularea funcțiilor ca expresii.
  Extindem intuiția de la expresii aritmetice pentru funcții.
- De exemplu, dacă în mod normal am scrie
  $$\text{Fie } f \text{ funcția } x \mapsto x^2. \text{ Atunci } A = f(5),$$
  în lambda calcul scriem doar
  $$A = (\lambda x.x^2)(5).$$
- Expresia $\lambda x.x^2$ reprezintă funcția care duce x în x^2
  (nu instrucțiunea/declarația că x este dus în x^2).
- Variabila x este locală/legată în termenul $\lambda x.x^2$
  De aceea, nu contează dacă am fi scris $\lambda y.y^2$

Funcții de nivel înalt

- Lambda calculul ne permite să lucrăm ușor cu funcții de nivel înalt (funcții ale căror intrări/ieșiri sunt tot funcții).
- De exemplu, operația $f \circ f$ este exprimată în lambda calcul prin
$$\lambda x.f(f(x))$$
iar operația $f \mapsto f \circ f$ prin
$$\lambda f.\lambda x.f(f(x))$$
- Evaluarea funcțiilor de nivel înalt poate deveni complexă.
De exemplu, expresia
$$((\lambda f.\lambda x.f(f(x)))(\lambda y.y^2))(5)$$
se evalueaza la 625.

Lambda calcul fără tipuri vs cu tipuri

Cateva exemple:

- Funcția identitate $f = \lambda x.x$ are tipul $X \to X$.
    - X poate să fie orice multime
    - contează doar ca domeniul și codomeniul să coincidă
- Funcția $g = \lambda f.\lambda x.f(f(x))$ are tipul $(X \to X) \to (X \to X)$
    - g duce orice funcție $f : X \to X$ intr-o funcție $g(f) : X \to X$

Lambda calcul fără tipuri vs cu tipuri

Permițând flexibilitate în alegerea domeniilor și a codomeniilor, putem manipula funcții în moduri surprinzătoare. De exemplu,

- Pentru funcția identitate $f = \lambda x.x$ avem $f(x) = x$, pentru orice x. În particular, putem lua $x = f$ si obținem

$$f(f) \simeq (\lambda x.x)(\lambda x.x) \simeq \lambda x.x \simeq f.$$

- Combinatorul $\omega = \lambda x.xx$ care pentru un x reprezintă funcția care aplică x lui x

$$\omega(\lambda y.y) \simeq (\lambda x.xx)(\lambda y.y) \simeq (\lambda y.y)(\lambda y.y) \simeq (\lambda y.y)$$

Ce reprezintă $\omega(\omega)$?

Lambda calcul fără tipuri vs cu tipuri

Permițând flexibilitate în alegerea domeniilor și a codomeniilor, putem manipula funcții în moduri surprinzătoare. De exemplu,

- Pentru funcția identitate $f = \lambda x.x$ avem $f(x) = x$, pentru orice x. În particular, putem lua $x = f$ si obținem

  $$f(f) \simeq (\lambda x.x)(\lambda x.x) \simeq \lambda x.x \simeq f.$$

- Combinatorul $\omega = \lambda x.xx$ care pentru un x reprezintă funcția care aplică x lui x

  $$\omega(\lambda y.y) \simeq (\lambda x.xx)(\lambda y.y) \simeq (\lambda y.y)(\lambda y.y) \simeq (\lambda y.y)$$

  Ce reprezintă $\omega(\omega)$?

  $$\omega(\omega) \simeq (\lambda x.xx)(\lambda x.xx) \simeq (\lambda x.xx)(\lambda x.xx)$$

Lambda calcul

- Lambda calcul fără tipuri
  - nu specificăm tipul niciunei expresii
  - nu specificăm domeniul/codomeniul funcțiilor
  - flexibilitate maximă, dar riscant deoarece putem ajunge în situații în care încercăm să aplicăm o funcție unui argument pe care nu îl poate procesa
- Lambda calcul cu tipuri simple
  - specificăm mereu tipul oricărei expresii
  - nu putem aplica funcții unui argument care are alt tip față de domeniul funcției
  - expresiile de forma $f(f)$ sunt eliminate, chiar dacă f este funcția identitate
- Lambda calcul cu tipuri polimorfice
  - o situație intermediară între cele două de mai sus
  - de exemplu, putem specifica că o expresie are tipul $X \rightarrow X$, dar fără a specifica cine este X

Calculabilitate

- Una din marile întrebări din anii 1930:

  Ce înseamnă că o funcție $f : \mathbb{N} \to \mathbb{N}$ *este calculabilă?*

- O definiție informală:

  ar trebui să existe o "metodă pe foaie" *(pen-and-paper)*
  care să îi permită unei persoane cu experiență
  să calculeze $f(n)$, pentru orice n.

- Conceptul de metodă "pen-and-paper" nu este ușor de formalizat

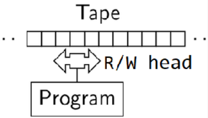
Definiții pentru Calculabilitate

1. Turing – a definit un calculator ideal numit mașina Turing și a postulat că o funcție este calculabilă ddacă poate fi calculată de o astfel de mașină.
2. Gödel – a definit clasa funcțiilor recursive și a postulat că o funcție este calculabilă ddacă este o funcție recursivă.
3. Church – a definit un limbaj de programare ideal numit lambda calcul și a postulat că o funcție este calculabilă ddacă poate fi scrisă ca un lambda termen.

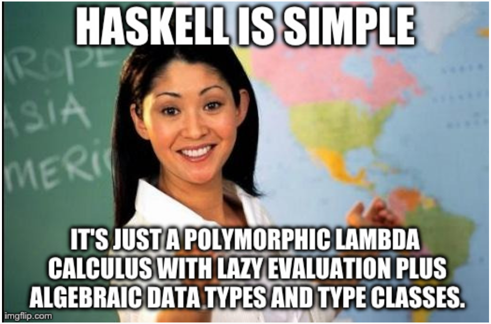
Teza Church-Turing

- Church, Kleene, Rosser și Turing au arătat că cele trei modele de calculabilitate sunt echivalente (definesc aceeași clasă de funcții calculabile).
- Dacă sunt sau nu echivalente cu noțiunea "intuitivă" de calculabilitate este o întrebare la care nu se poate răspunde, deoarece nu avem o definiție pentru "calculabilitate intuitivă".
- Faptul că cele trei modele coincid cu noțiunea intuitivă de calculabilitate se numește teza Church-Turing.

Syntax	Name
x	Variable
(λx.M)	Abstraction
(M N)	Application



Lambda calcul în informatică



- Lambda calcul este un limbaj de programare ideal.
- Probabil cel mai simplu limbaj de programare Turing complet.
- Toate limbajele de programare funcțională sunt extensii ale lambda calculului cu diferite caracteristici (tipuri de date, efecte laterale etc)

Proofs-as-programs

- Ce este o demonstrație?
    - Logica clasică: plecând de la niște presupuneri, este suficient să ajungi la o contradicție
    - Logica constructivistă: pentru a arata ca un obiect există, trebuie să îl construim explicit.
- Legătura dintre lambda calcul și logica constructivistă este dată de paradigma *proofs-as-programs*.
    - o demonstrație trebuie să fie o "construcție", un program
    - lambda calculul este o notație pentru astfel de programe

Quiz time!

https://tinyurl.com/C01-Quiz1

Pe săptămâna viitoare!