

# Fundamentele limbajelor de programare

C01

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

# Organizare

---

- 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

- Moodle

- Teams

<https://tinyurl.com/FLP2023-Teams>

- Suporturile de curs si laborator

<https://tinyurl.com/FLP2023-Materials>

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

## 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

- 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 asemănătoare cu cele din quiz-urile de la curs
- materiale ajutătoare: suporturile de curs și de laborator



- 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

- 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]  
ε ψιλόν

Ζζ

ZETA [dz]  
ζῆτα

Ηη

ETA [ɛː]  
ἦτα

Θθ

THETA [tʰ]  
θῆτα

Ιι

IOTA [i]  
ιῶτα

Κκ

KAPPA [k]  
κάππα

Λλ

LAMBDA [l]  
λάμβδα

Μμ

MU [m]  
μῦ

Νν

NU [n]  
νῦ

Ξξ

XI [ks]  
ξεῖ

Οο

OMICRON [o]  
ὀ μικρόν

Ππ

PI [p]  
πεῖ

Ρρ

RHO [r]  
ῥῶ

Σσς

SIGMA [s]  
σίγμα

Ττ

TAU [t]  
ταῦ

Υυ

UPSILON [u]  
ὕ ψιλόν

Φφ

PHI [pʰ]  
φεῖ

Χχ

CHI [kʰ]  
χεῖ

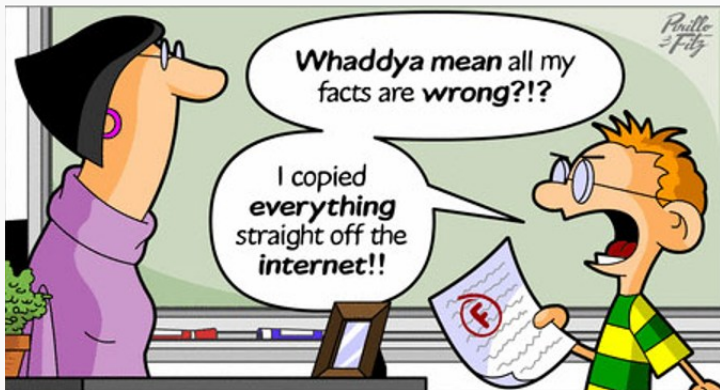
Ψψ

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?

- În matematica modernă, avem "funcții prin grafice":
  - orice funcție  $f$  are un domeniu  $X$  și un codomeniu  $Y$  fixate, și
  - orice funcție  $f : X \rightarrow Y$  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  $f, g : X \rightarrow Y$  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 egală cu  $g$  de mai jos?

$$g(x) = (x - 1)(x + 1).$$

- Dacă 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  $h : X \rightarrow X$ , 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 ( $x, y, z, \dots$ )
  - numere ( $1, 2, 3, \dots$ )
  - 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

fie  $w = x + y$ , apoi fie  $u = z^2$ , 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

Fie  $f$  funcția  $x \mapsto x^2$ . 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 evaluează la 625.

# Lambda calcul fără tipuri vs cu tipuri

Cateva exemple:

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

## 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  $x = f$  și 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  $x = f$  și 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 polimorifice

- 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  $X$



- Una din marile întrebări din anii 1930:

*Ce înseamnă că o funcție  $f : \mathbb{N} \rightarrow \mathbb{N}$  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  $n$ .

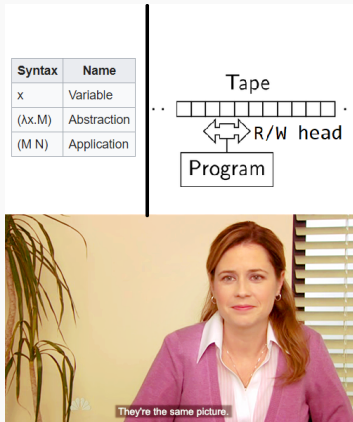
- Conceptul de metodă "pen-and-paper" nu este ușor de formalizat

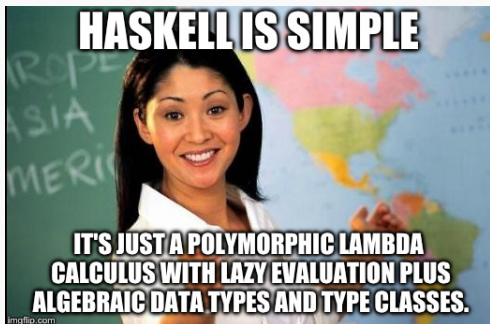
## Definiții pentru Calculabilitate

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**.

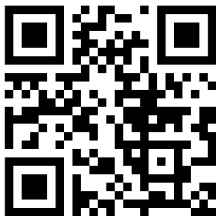




- Lambda calcul este un limbaj de programare ideal.
- Probabil cel mai simplu limbaj de programare Turing complet.
- Toate limbajele de programare funcțională sunt extensii ale lambda calculului cu diferite caracteristici (tipuri de date, efecte laterale etc)

- 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 notăție pentru astfel de programe

Quiz time!



<https://tinyurl.com/C01-Quiz1>

**Pe săptămâna viitoare!**

# Fundamentele limbajelor de programare

C02

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB



## **Lambda calcul - elemente de bază**

---

- Un model de calculabilitate
- Limbajele de programare funcțională sunt extensii ale sale
- Un limbaj formal
- Expresiile din acest limbaj se numesc **lambda termeni**
- Vom defini reguli pentru a îi manipula

# Lambda termenii

Fie  $V$  o mulțime infinită de variabile, notate  $x, y, z, \dots$

Mulțimea **lambda termenilor** este dată de următoarea formă BNF:

$$\begin{array}{lcl} \text{lambda termen} & = & \text{variabilă} \\ & | & \text{aplicare} \\ & | & \text{abstractizare} \end{array}$$
$$M, N ::= x \mid (MN) \mid (\lambda x.M)$$

## Example

- $x, y, z$
- $(x\ y), (y\ x), (x\ (y\ x))$
- $(\lambda x.x), \lambda x.(x\ y), \lambda z.(x\ y)$
- $((\lambda x.x)\ y), ((\lambda x.(x\ z))\ y)$
- $(\lambda f.(\lambda x.(f\ (f\ x))))$
- $(\lambda x.x)\ (\lambda x.x)$

# Funcții anonime în Haskell

lambda termen = variabilă  
| aplicare  
| abstractizare

$M, N ::= x \mid (M\ N) \mid (\lambda x.M)$

În Haskell, `\` e folosit în locul simbolului  $\lambda$  și `->` în locul punctului:

$\lambda x.x * x$  este `\x -> x * x`

$\lambda x.x > 0$  este `\x -> x > 0`

## Lambda termeni - definiție alternativă

Fie  $V$  o mulțime infinită de variabile, notate  $x, y, z, \dots$

Fie  $A$  un alfabet format din elementele din  $V$ , și simbolurile speciale  $"(", ")", "\lambda"$  și  $"."$

Fie  $A^*$  mulțimea tuturor cuvintelor finite pentru alfabetul  $A$ .

Mulțimea **lambda termenilor** este cea mai mică submulțime  $\Lambda \subseteq A^*$  astfel încât:

[Variabilă]  $V \subseteq \Lambda$

[Aplicare] dacă  $M, N \in \Lambda$  atunci  $(M N) \in \Lambda$

[Abstractizare] dacă  $x \in V$  și  $M \in \Lambda$  atunci  $(\lambda x.M) \in \Lambda$

# Convenții

- Se elimină parantezele exterioare
- Aplicarea este asociativă la stânga
  - $MNP$  înseamnă  $(MN)P$
  - $fxyz$  înseamnă  $((fx)y)z$
- Corpul abstractizării (partea de după punct) se extinde la dreapta cât se poate
  - $\lambda x.MN$  înseamnă  $\lambda x.(MN)$ , nu  $(\lambda x.M)N$
- Mai mulți  $\lambda$  pot fi comprimați
  - $\lambda xyz.M$  este o abreviere pentru  $\lambda x.\lambda y.\lambda z.M$

Aceste convenții nu afectează definiția lambda termenilor.

**Exercițiu.** Scrieți termenii de mai jos cu cât mai puține paranteze și folosind convențiile de mai sus, fără a schimba sensul termenilor:

1.  $(\lambda x.(\lambda y.(\lambda z.((x\ z)(y\ z)))))$
2.  $((((a\ b)(c\ d))((e\ f)(g\ h))))$

**Exercițiu.** Adăugați parantezele în termenii de mai jos astfel încât să nu le schimbați sensul:

1.  $x\ x\ x\ x$
2.  $\lambda x.x\ \lambda y.y$

## Exerciții

**Exercițiu.** Scrieți termenii de mai jos cu cât mai puține paranteze și folosind convențiile de mai sus, fără a schimba sensul termenilor:

1.  $(\lambda x.(\lambda y.(\lambda z.((x\ z)(y\ z)))))$       Corect:  $\lambda xyz.x\ z\ (y\ z)$
2.  $((((a\ b)\ (c\ d))\ ((e\ f)\ (g\ h))))$

**Exercițiu.** Adăugați parantezele în termenii de mai jos astfel încât să nu le schimbați sensul:

1.  $x\ x\ x\ x$
2.  $\lambda x.x\ \lambda y.y$



## Exerciții

**Exercițiu.** Scrieți termenii de mai jos cu cât mai puține paranteze și folosind convențiile de mai sus, fără a schimba sensul termenilor:

1.  $(\lambda x.(\lambda y.(\lambda z.((x\ z)(y\ z)))))$  Corect:  $\lambda xyz.x\ z\ (y\ z)$

2.  $((((a\ b)\ (c\ d))\ ((e\ f)\ (g\ h))))$  Corect:  $a\ b\ (c\ d)\ (e\ f\ (g\ h))$

**Exercițiu.** Adăugați parantezele în termenii de mai jos astfel încât să nu le schimbați sensul:

1.  $x\ x\ x\ x$

2.  $\lambda x.x\ \lambda y.y$

## Exerciții

**Exercițiu.** Scrieți termenii de mai jos cu cât mai puține paranteze și folosind convențiile de mai sus, fără a schimba sensul termenilor:

1.  $(\lambda x.(\lambda y.(\lambda z.((x\ z)(y\ z)))))$  Corect:  $\lambda xyz.x\ z\ (y\ z)$

2.  $((((a\ b)\ (c\ d))\ ((e\ f)\ (g\ h))))$  Corect:  $a\ b\ (c\ d)\ (e\ f\ (g\ h))$

**Exercițiu.** Adăugați parantezele în termenii de mai jos astfel încât să nu le schimbați sensul:

1.  $x\ x\ x\ x$  Corect:  $((((x\ x)\ x)\ x))$

2.  $\lambda x.x\ \lambda y.y$

## Exerciții

**Exercițiu.** Scrieți termenii de mai jos cu cât mai puține paranteze și folosind convențiile de mai sus, fără a schimba sensul termenilor:

1.  $(\lambda x.(\lambda y.(\lambda z.((x\ z)(y\ z)))))$  Corect:  $\lambda xyz.x\ z\ (y\ z)$

2.  $((((a\ b)\ (c\ d))\ ((e\ f)\ (g\ h))))$  Corect:  $a\ b\ (c\ d)\ (e\ f\ (g\ h))$

**Exercițiu.** Adăugați parantezele în termenii de mai jos astfel încât să nu le schimbați sensul:

1.  $x\ x\ x\ x$  Corect:  $((((x\ x)\ x)\ x))$

2.  $\lambda x.x\ \lambda y.y$  Corect:  $(\lambda x.(x\ (\lambda y.y)))$

## Variabile libere și variabile legate

- $\lambda\_.$  se numește operator **de legare** (*binder*)
- $x$  din  $\lambda x. \_$  se numește variabilă **de legare** (*binding*)
- $N$  din  $\lambda x. N$  se numește **domeniul** (*scope*) de legare a lui  $x$
- toate aparițiile lui  $x$  în  $N$  sunt legate
- O apariție care nu este legată se numește **liberă**.
- Un termen fără variabile libere se numește **închis** (*closed*).
- Un termen închis se mai numește și **combinator**.

De exemplu, în termenul

$$M \equiv (\lambda x. xy) (\lambda y. yz)$$

- $x$  este legată
- $z$  este liberă
- $y$  are și o apariție legată, și una liberă
- mulțimea variabilelor libere ale lui  $M$  este  $\{y, z\}$

## Variabile libere

Mulțimea **variabilelor libere** dintr-un termen  $M$  este notată  $FV(M)$  și este definită formal prin:

$$\begin{aligned}FV(x) &= \{x\} \\FV(MN) &= FV(M) \cup FV(N) \\FV(\lambda x.M) &= FV(M) \setminus \{x\}\end{aligned}$$

Exemplu de definiție recursivă pe termeni. Adică în definiția lui  $FV(M)$  am presupus că am definit deja  $FV(N)$  pentru toți subtermenii lui  $M$ .

### Example

- $FV(\lambda x.x y) = FV(x y) \setminus \{x\} = (FV(x) \cup FV(y)) \setminus \{x\}$   
 $= (\{x\} \cup \{y\}) \setminus \{x\} = \{y\}$
- $FV(x \lambda x.x y) = \{x, y\}$

## Redenumire de variabile

Ce înseamnă să redenumim o variabilă într-un termen?

Dacă  $x, y$  sunt variabile și  $M$  este un termen,  $M\langle y/x \rangle$  este rezultatul obținut după redenumirea lui  $x$  cu  $y$  în  $M$ .

$$x\langle y/x \rangle \equiv y,$$

$$z\langle y/x \rangle \equiv z, \quad \text{dacă } x \neq z$$

$$(MN)\langle y/x \rangle \equiv (M\langle y/x \rangle)(N\langle y/x \rangle)$$

$$(\lambda x.M)\langle y/x \rangle \equiv \lambda y.(M\langle y/x \rangle)$$

$$(\lambda z.M)\langle y/x \rangle \equiv \lambda z.(M\langle y/x \rangle), \quad \text{dacă } x \neq z$$

Observați că acest tip de redenumire înlocuiește toate aparițiile lui  $x$  cu  $y$ , indiferent dacă este liberă, legată, sau de legare.

Se folosește doar în cazuri în care  $y$  nu apare deja în  $M$ .

Ce înseamnă că doi termeni sunt egali,  
**modulo redenumire de variabile legate?**

Definim  $\alpha$ -echivalența ca fiind cea mai mică relație de congruență  $=_\alpha$  pe mulțimea lambda termenilor, astfel încât pentru orice termen  $M$  și orice variabilă  $y$  care nu apare în  $M$ , avem

$$\lambda x.M =_\alpha \lambda y.(M\langle y/x \rangle)$$

## $\alpha$ -echivalență

$\alpha$ -echivalența  $=_{\alpha}$  este cea mai mică relație pe lambda termeni care satisface regulile:

$(refl)$	$\frac{}{M = M}$	$(cong)$	$\frac{M = M' \quad N = N'}{MN = M'N'}$
$(symm)$	$\frac{M = N}{N = M}$	$(\xi)$	$\frac{M = M'}{\lambda x.M = \lambda x.M'}$
$(trans)$	$\frac{M = N \quad N = P}{M = P}$	$(\alpha)$	$\frac{y \notin M}{\lambda x.M = \lambda y.(M\{y/x\})}$

Convenția Barendregt:

variabilele legate sunt redenumite pentru a fi distincte.



Vrem să substituim variabile cu lambda termeni.

$M[N/x]$  este rezultatul obținut după înlocuirea lui  $x$  cu  $N$  în  $M$ .

Trebuie să fim atenți la următoarele cazuri:

1. Vrem să înlocuim doar variabile libere.

Numele variabilelor legate este considerat imaterial, și nu ar trebui să afecteze rezultatul substituției.

De exemplu,  $x(\lambda xy.x)[N/x]$  ar trebui să fie  $N(\lambda xy.x)$ ,  
nu  $N(\lambda xy.N)$  sau  $N(\lambda Ny.N)$ .

## 2. Nu vrem să legăm variabile libere neintenționat.

De exemplu, fie  $M \equiv \lambda x. y x$  și  $N \equiv \lambda z. x z$ .

Variabila  $x$  este legată în  $M$  și liberă în  $N$ .

Ce ar trebui să obținem dacă am substitui  $y$  cu  $N$  în  $M$ ?

Naiv, ne-am gândi la

$$M[N/y] = (\lambda x. y x)[N/y] = \lambda x. N x = \lambda x. (\lambda z. x z) x.$$

Totuși, nu este ceea ce am vrea să obținem, deoarece  $x$  este liber în  $N$ , iar în timpul "substituției" a devenit legată.

Trebuie să luăm în calcul că  $x$ -ul legat din  $M$  nu este  $x$ -ul liber din  $N$ , și de aceea **redenumim variabilele legate** înainte de substituție.

$$M[N/y] = (\lambda x'. y x')[N/y] = \lambda x'. N x' = \lambda x'. (\lambda z. x z) x'.$$

# Substituții

**Substituția** aparițiilor libere ale lui  $x$  cu  $N$  în  $M$ , notată cu  $M[N/x]$ , este definită prin:

$$\begin{array}{lll} x[N/x] & \equiv & N \\ y[N/x] & \equiv & y \quad \text{dacă } x \neq y \\ (MP)[N/x] & \equiv & (M[N/x])(P[N/x]) \\ (\lambda x.M)[N/x] & \equiv & \lambda x.M \\ (\lambda y.M)[N/x] & \equiv & \lambda y.(M[N/x]) \quad \text{dacă } x \neq y \text{ și } y \notin FV(N) \\ (\lambda y.M)[N/x] & \equiv & \lambda y'.(M[y'/y][N/x]) \quad \text{dacă } x \neq y, y \in FV(N) \\ & & \text{și } y' \text{ variabilă nouă} \end{array}$$

Deoarece nu specificăm ce variabilă nouă alegem, spunem că substituția este bine-definită modulo  $\alpha$ -echivalențe.

**Exercițiu.** Calculați următoarele substituții:

1.  $(\lambda z.x)[y/x]$

2.  $(\lambda y.x)[y/x]$

3.  $(\lambda y.x)[(\lambda z.z\ w)/x]$

**Exercițiu.** Calculați următoarele substituții:

1.  $(\lambda z.x)[y/x]$

Corect:  $\lambda z.y$

2.  $(\lambda y.x)[y/x]$

3.  $(\lambda y.x)[(\lambda z.z\ w)/x]$

**Exercițiu.** Calculați următoarele substituții:

1.  $(\lambda z.x)[y/x]$

Corect:  $\lambda z.y$

2.  $(\lambda y.x)[y/x]$

Corect:  $\lambda y'.y$ , Greșit:  $\lambda y.y$

3.  $(\lambda y.x)[(\lambda z.z\ w)/x]$

**Exercițiu.** Calculați următoarele substituții:

1.  $(\lambda z.x)[y/x]$

Corect:  $\lambda z.y$

2.  $(\lambda y.x)[y/x]$

Corect:  $\lambda y'.y$ , Greșit:  $\lambda y.y$

3.  $(\lambda y.x)[(\lambda z.z\ w)/x]$

Corect:  $\lambda yz.zw$

Quiz time!



<https://tinyurl.com/C02-Quiz1>



**Pe săptămâna viitoare!**

# Fundamentele limbajelor de programare

C03

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## Lambda calcul - $\beta$ -reducții

---

**Convenție.** Spunem că doi termeni sunt egali, notat  $M = N$ , dacă sunt  $\alpha$ -echivalenți.

- $\beta$ -reducție = procesul de a evalua lambda termeni prin "pasarea de argumente funcțiilor"
- $\beta$ -redex = un termen de forma  $(\lambda x.M) N$
- redusul unui redex  $(\lambda x.M) N$  este  $M[N/x]$
- reducem lambda termeni prin găsirea unui subtermen care este redex, și apoi înlocuirea acelu redex cu redusul său
- repetăm acest proces de câte ori putem, până nu mai sunt redex-uri
- formă normală = un lambda termen fără redex-uri

Un pas de  $\beta$ -reducție  $\rightarrow_\beta$  este cea mai mică relație pe lambda termeni care satisface regulile:

$$\begin{array}{ll} (\beta) & \overline{(\lambda x.M)N \rightarrow_\beta M[N/x]} \\ (cong_1) & \frac{M \rightarrow_\beta M'}{MN \rightarrow_\beta M'N} \\ (cong_2) & \frac{N \rightarrow_\beta N'}{MN \rightarrow_\beta MN'} \\ (\xi) & \frac{M \rightarrow_\beta M'}{\lambda x.M \rightarrow_\beta \lambda x.M'} \end{array}$$

La fiecare pas, subliniem redexul ales în procesul de  $\beta$ -reducție.

$$\begin{aligned}(\lambda x.y) (\underline{(\lambda z.zz) (\lambda w.w)}) &\rightarrow_{\beta} (\lambda x.y) ((z\ z)[\lambda w.w/z]) \\&\equiv (\lambda x.y) ((z[\lambda w.w/z]) (z[\lambda w.w/z])) \\&\equiv (\lambda x.y) (\underline{(\lambda w.w) (\lambda w.w)}) \\&\rightarrow_{\beta} \underline{(\lambda x.y) (\lambda w.w)} \\&\rightarrow_{\beta} y\end{aligned}$$

Ultimul termen nu mai are redex-uri, deci este în formă normală.

$$\begin{aligned}(\lambda x.y) ((\lambda z.zz) (\lambda w.w)) &\rightarrow_{\beta} (\lambda x.y) ((\lambda w.w) (\lambda w.w)) \\&\rightarrow_{\beta} (\lambda x.y) (\lambda w.w) \\&\rightarrow_{\beta} y\end{aligned}$$

$$\begin{aligned}(\lambda x.y) ((\lambda z.zz) (\lambda w.w)) &\rightarrow_{\beta} y[(\lambda z.zz) (\lambda w.w)/x] \\&\equiv y\end{aligned}$$

Observăm că:

- reducerea unui redex poate crea noi redex-uri
- reducerea unui redex poate șterge alte redex-uri
- numărul de pași necesari până a atinge o formă normală poate varia, în funcție de ordinea în care sunt reduse redex-urile
- rezultatul final pare că nu a depins de alegerea redex-urilor

Totuși, există lambda termeni care nu pot fi reduși la o  $\beta$ -formă normală (evaluarea nu se termină).

$$\begin{array}{ccc} \underline{(\lambda x.x x) (\lambda x.x x)} & \rightarrow_{\beta} & (\lambda x.x x) (\lambda x.x x) \\ & \rightarrow_{\beta} & \dots \end{array}$$

Observați că lungimea unui termen nu trebuie să scadă în procesul de  $\beta$ -reducție; poate crește sau rămâne neschimbat.



Există lambda termeni care deși pot fi reduși la o formă normală, pot să nu o atingă niciodată.

$$\begin{array}{ccc} \underline{(\lambda xy.y) ((\lambda x.x x) (\lambda x.x x)) (\lambda z.z)} & \rightarrow_{\beta} & \underline{(\lambda y.y) (\lambda x.x)} \\ & \rightarrow_{\beta} & \lambda x.x \end{array}$$

$$(\lambda xy.y) (\underline{(\lambda x.x x) (\lambda x.x x)}) (\lambda z.z) \rightarrow_{\beta} (\lambda xy.y) ((\lambda x.x x) (\lambda x.x x)) (\lambda z.z)$$

Contează **strategia de evaluare**.

## $\beta$ -formă normală

Notăm cu  $M \rightarrow_{\beta}^* M'$  faptul că  $M$  poate fi  $\beta$ -redus până la  $M'$  în 0 sau mai mulți pași (închiderea reflexivă și tranzitivă a relației  $\rightarrow_{\beta}$ ).

$M$  este **slab normalizabil** (*weakly normalising*) dacă există  $N$  în formă normală astfel încât  $M \rightarrow_{\beta}^* N$ .

$M$  este **puternic normalizabil** (*strong normalising*) dacă nu există reduceri infinite care încep din  $M$ .

Orice termen puternic normalizabil este și slab normalizabil.

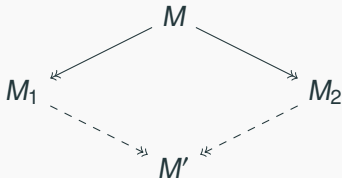
### Example

$(\lambda x.y)((\lambda z.zz)(\lambda w.w))$  este **puternic normalizabil**.

$(\lambda xy.y)((\lambda x.x x)(\lambda x.x x))(\lambda z.z)$  este **slab normalizabil**,  
dar **nu puternic normalizabil**.

# Confluența $\beta$ -reducției

**Teorema Church-Rosser.** Dacă  $M \rightarrow_{\beta} M_1$  și  $M \rightarrow_{\beta} M_2$  atunci există  $M'$  astfel încât  $M_1 \rightarrow_{\beta} M'$  și  $M_2 \rightarrow_{\beta} M'$ .



**Consecință.** Un lambda termen are cel mult o  $\beta$ -formă normală (modulo  $\alpha$ -echivalență).

**Exercițiu.** Verificați dacă termenii de mai jos pot fi aduși la o  $\beta$ -formă normală:

1.  $(\lambda x.x) M$
2.  $(\lambda xy.x) M N$
3.  $(\lambda x.x x) (\lambda y.y y y)$

**Exercițiu.** Verificați dacă termenii de mai jos pot fi aduși la o  $\beta$ -formă normală:

1.  $(\lambda x.x) M$  Corect:  $M$

2.  $(\lambda xy.x) M N$  Corect:  $M$

3.  $(\lambda x.x x) (\lambda y.y y y)$  Corect:  $(\lambda y.y y y) (\lambda y.y y y) (\lambda y.y y y) \dots$

## **Strategii de evaluare**

---

De cele mai multe ori, există mai mulți pași de  $\beta$ -reducție care pot fi aplicați unui termen. Cum alegem ordinea? Contează ordinea?

O **strategie de evaluare** ne spune în ce ordine să facem pașii de reducere.

Lambda calculul nu specifică o strategie de evaluare, fiind **nedeterminist**. O strategie de evaluare este necesară în limbaje de programare reale pentru a rezolva nedeterminismul.

## Strategia normală (normal order)

Strategia normală = *leftmost-outermost*

(alegem redex-ul cel mai din stânga și apoi cel mai din exterior)

- dacă  $M_1$  și  $M_2$  sunt redex-uri și  $M_1$  este un subtermen al lui  $M_2$ , atunci  $M_1$  **nu** va fi următorul redex ales
- printre redex-urile care nu sunt subtermeni ai altor redex-uri (și sunt incomparabili față de relația de subtermen), îl alegem pe cel mai din stânga.

Dacă un termen are o formă normală, atunci strategia normală va converge la ea.

$$\begin{array}{ccc} \frac{(\lambda xy.y) ((\lambda x.x x) (\lambda x.x x)) (\lambda z.z)}{} & \rightarrow_{\beta} & \frac{(\lambda y.y) (\lambda x.x)}{\lambda x.x} \\ & \rightarrow_{\beta} & \end{array}$$



## Strategia aplicativă (applicative order)

Strategia aplicativă = *leftmost-innermost*

(alegem redex-ul cel mai din stânga și apoi cel mai din interior)

- dacă  $M_1$  și  $M_2$  sunt redex-uri și  $M_1$  este un subtermen al lui  $M_2$ , atunci  $M_2$  **nu** va fi următorul redex ales
- printre redex-urile care nu sunt subtermeni ai altor redex-uri (și sunt incomparabili față de relația de subtermen), îl alegem pe cel mai din stânga.

$$(\lambda xy.y) (\underline{(\lambda x.x x) (\lambda x.x x)}) (\lambda z.z) \rightarrow_{\beta} (\lambda xy.y) ((\lambda x.x x) (\lambda x.x x)) (\lambda z.z)$$

## Strategii în programare funcțională

În limbaje de programare funcțională, în general, reducerile din corpul unei  $\lambda$ -abstractizări nu sunt efectuate (deși anumite compilatoare optimizate pot face astfel de reduceri în unele cazuri).

Strategia *call-by-name* (CBN) = strategia normală fără a face reduceri în corpul unei  $\lambda$ -abstractizări

Strategia *call-by-value* (CBV) = strategia aplicativă fără a face reduceri în corpul unei  $\lambda$ -abstractizări

Majoritatea limbajelor de programare funcțională folosesc CBV, excepție făcând Haskell.

## CBN vs CBV

O **valoare** este un  $\lambda$ -term pentru care nu există  $\beta$ -reducții date de strategia de evaluare considerată.

De exemplu,  $\lambda x.x$  este mereu o valoare, dar  $(\lambda x.x) 1$  nu este.

Sub **CBV**, funcțiile pot fi apelate doar prin valori (argumentele trebuie să fie complet evaluate). Astfel, putem face  $\beta$ -reducția  $(\lambda x.M) N \rightarrow_{\beta} M[N/x]$  doar dacă  $N$  este valoare.

Sub **CBN**, amânăm evaluarea argumentelor cât mai mult posibil, făcând reducții de la stânga la dreapta în expresie. Aceasta este strategia folosită în Haskell.

CBN este o formă de evaluare leneșă (*lazy evaluation*): argumentele funcțiilor sunt evaluate doar când sunt necesare.

## Example

Considerăm 3 și *succ* primitive.

Strategia CBV:

$$\begin{aligned}(\lambda x.succ\ x)\ ((\lambda y.succ\ y)\ 3) &\rightarrow_{\beta} (\lambda x.succ\ x)\ (succ\ 3) \\&\rightarrow (\lambda x.succ\ x)\ 4 \\&\rightarrow_{\beta} succ\ 4 \\&\rightarrow 5\end{aligned}$$

Strategia CBN:

$$\begin{aligned}(\lambda x.succ\ x)\ ((\lambda y.succ\ y)\ 3) &\rightarrow_{\beta} succ\ ((\lambda y.succ\ y)\ 3) \\&\rightarrow_{\beta} succ\ (succ\ 3) \\&\rightarrow succ\ 4 \\&\rightarrow 5\end{aligned}$$

Quiz time!



<https://tinyurl.com/C03-Quiz1>

**Pe săptămâna viitoare!**

# Fundamentele limbajelor de programare

C04

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## Expresivitatea $\lambda$ -calculului

---



Deși lambda calculul constă doar în  $\lambda$ -termeni, putem reprezenta și manipula tipuri de date comune.

Vom vedea cum putem reprezenta:

- valori booleene
- numere naturale

## Booleeni

---

Vrem să definim  $\lambda$ -termeni care să reprezinte constantele booleene.

Sunt mai multe modalități, una dintre ele fiind:

- $\mathbf{T} \triangleq \lambda xy.x$  (dintre cele două alternative o alege pe prima)
- $\mathbf{F} \triangleq \lambda xy.y$  (dintre cele două alternative o alege pe a doua)

$$\mathbf{T} \triangleq \lambda xy.x \qquad \mathbf{F} \triangleq \lambda xy.y$$

Acum am vrea să definim un test condiționat **if**.

Am vrea ca **if** să ia trei argumente  $b, t, f$ , unde  $b$  este o valoare booleană, iar  $t, f$  sunt  $\lambda$ -termeni oarecare.

Funcția ar trebui să returneze  $t$  dacă  $b = \text{true}$  și  $f$  dacă  $b = \text{false}$

$$\mathbf{if} = \lambda btf. \begin{cases} t, & \text{if } b = \text{true}, \\ f, & \text{if } b = \text{false}. \end{cases}$$

Deoarece  $\mathbf{T} t f \rightarrow_{\beta} t$  și  $\mathbf{F} t f \rightarrow_{\beta} f$ , **if** trebuie doar să aplice argumentul său boolean celorlalte argumente:

$$\mathbf{if} \triangleq \lambda btf.b t f$$

# Booleeni

$$\mathbf{T} \triangleq \lambda xy.x$$

$$\mathbf{F} \triangleq \lambda xy.y$$

$$\mathbf{if} \triangleq \lambda btf.b \ t \ f$$

Celelalte operații booleene pot fi definite folosind **if**:

$$\mathbf{and} \triangleq \lambda b_1 b_2. \mathbf{if} \ b_1 \ b_2 \ \mathbf{F}$$

$$\mathbf{or} \triangleq \lambda b_1 b_2. \mathbf{if} \ b_1 \ \mathbf{T} \ b_2$$

$$\mathbf{not} \triangleq \lambda b_1. \mathbf{if} \ b_1 \ \mathbf{F} \ \mathbf{T}$$

Observați că aceste operații lucrează corect doar dacă primesc ca argumente valori booleene.

Nu există nicio garanție să se comporte rezonabil pe orice alți  $\lambda$ -termeni.

Folosind lambda calcul fără tipuri, avem *garbage in, garbage out*.

Codările nu sunt unice. De exemplu, pentru **and** am fi putut folosi codările  $\lambda b_1 b_2. b_2 \ b_1 \ b_2$  sau  $\lambda b_1 b_2. b_1 \ b_2 \ \mathbf{F}$ .

$\mathbf{T} \triangleq \lambda xy.x$      $\mathbf{F} \triangleq \lambda xy.y$      $\mathbf{if} \triangleq \lambda btf.b\ t\ f$

$\mathbf{and} \triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ b_2\ \mathbf{F}$

$\mathbf{or} \triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ \mathbf{T}\ b_2$

$\mathbf{not} \triangleq \lambda b_1.\mathbf{if}\ b_1\ \mathbf{F}\ \mathbf{T}$

**Exercițiu.** Aduceți la o formă normală următorii termenii:

- **and TF**
- **or FT**
- **not T**

$\mathbf{T} \triangleq \lambda xy.x$      $\mathbf{F} \triangleq \lambda xy.y$      $\mathbf{if} \triangleq \lambda btf.b \ t \ f$

$\mathbf{and} \triangleq \lambda b_1 b_2.\mathbf{if} \ b_1 \ b_2 \ \mathbf{F}$

$\mathbf{or} \triangleq \lambda b_1 b_2.\mathbf{if} \ b_1 \ \mathbf{T} \ b_2$

$\mathbf{not} \triangleq \lambda b_1.\mathbf{if} \ b_1 \ \mathbf{F} \ \mathbf{T}$

**Soluții:**

$\mathbf{and} \ \mathbf{TF} = (\lambda b_1 b_2.\mathbf{if} \ b_1 \ b_2 \ \mathbf{F}) \ \mathbf{TF} \rightarrow_{\beta} \mathbf{if} \ \mathbf{T} \ \mathbf{F} \ \mathbf{F} = (\lambda btf.b \ t \ f) \ \mathbf{T} \ \mathbf{F} \ \mathbf{F}$   
 $\rightarrow_{\beta} \mathbf{T} \ \mathbf{F} \ \mathbf{F} = (\lambda xy.x) \ \mathbf{F} \ \mathbf{F} \rightarrow_{\beta} \mathbf{F}$

$\mathbf{or} \ \mathbf{FT} = (\lambda b_1 b_2.\mathbf{if} \ b_1 \ \mathbf{T} \ b_2) \ \mathbf{FT} \rightarrow_{\beta} \mathbf{if} \ \mathbf{F} \ \mathbf{T} \ \mathbf{T} = (\lambda btf.b \ t \ f) \ \mathbf{F} \ \mathbf{T} \ \mathbf{T}$   
 $\rightarrow_{\beta} \mathbf{F} \ \mathbf{T} \ \mathbf{T} = (\lambda xy.y) \ \mathbf{T} \ \mathbf{T} \rightarrow_{\beta} \mathbf{T}$

$\mathbf{not} \ \mathbf{T} = (\lambda b_1.\mathbf{if} \ b_1 \ \mathbf{F} \ \mathbf{T}) \ \mathbf{T} \rightarrow_{\beta} \mathbf{if} \ \mathbf{T} \ \mathbf{F} \ \mathbf{T} = (\lambda btf.b \ t \ f) \ \mathbf{T} \ \mathbf{F} \ \mathbf{T}$   
 $\rightarrow_{\beta} \mathbf{T} \ \mathbf{F} \ \mathbf{T} = (\lambda xy.x) \ \mathbf{F} \ \mathbf{T} \rightarrow_{\beta} \mathbf{F}$

## Numere naturale

---



## Numere naturale

Vom reprezenta numerele naturale  $\mathbb{N}$  folosind **numeralii Church**.

Numeralul Church pentru numărul  $n \in \mathbb{N}$  este notat  $\bar{n}$ .

Numeralul Church  $\bar{n}$  este  $\lambda$ -termenul  $\lambda f x. f^n x$ , unde  $f^n$  reprezintă compunerea lui  $f$  cu ea însăși de  $n$  ori:

$$\begin{aligned}\bar{0} &\triangleq \lambda f x. f^0 x &= \lambda f x. x \\ \bar{1} &\triangleq \lambda f x. f^1 x &= \lambda f x. f x \\ \bar{2} &\triangleq \lambda f x. f^2 x &= \lambda f x. f (f x) \\ \bar{3} &\triangleq \lambda f x. f^3 x &= \lambda f x. f (f (f x)) \\ &\vdots \\ \bar{n} &\triangleq \lambda f x. f^n x &= \lambda f x. \underbrace{f(f(\dots(f x) \dots))}_n\end{aligned}$$

## Numere naturale

$$\bar{n} \triangleq \lambda fx.f^n x$$

Acum putem defini funcția **succesor** prin

$$\mathbf{Succ} \triangleq \lambda nfx.f(n f x)$$

Observați că **Succ** pe argumentul  $\bar{n}$  returnează o funcție care primește ca argument o funcție  $f$ , îi aplică  $\bar{n}$  pentru a obține compunerea de  $n$  ori a lui  $f$  cu ea însăși, apoi aplică iar  $f$  pentru a obține compunerea de  $n + 1$  ori a lui  $f$  cu ea însăși.

$$\begin{aligned}\mathbf{Succ} \bar{n} &= (\lambda nfx.f(n f x)) \bar{n} \\ &\rightarrow_{\beta} \lambda fx.f(\bar{n} f x) \\ &\rightarrow_{\beta} \lambda fx.f(f^n x) \\ &= \lambda fx.f^{n+1} x \\ &= \overline{n+1}\end{aligned}$$

## Numere naturale

$$\bar{n} \triangleq \lambda fx.f^n x \qquad \mathbf{Succ} \triangleq \lambda nfx.f (n f x)$$

Putem face operații aritmetice de bază cu numeralii Church.

Pentru **adunare**, putem defini

$$\mathbf{add} \triangleq \lambda mnfx.m f (n f x)$$

Pentru argumentele  $\bar{m}$  și  $\bar{n}$ , obținem:

$$\begin{aligned} \mathbf{add} \bar{m} \bar{n} &= (\lambda mnfx.m f (n f x)) \bar{m} \bar{n} \\ &\rightarrow_{\beta} \lambda fx.\bar{m} f (\bar{n} f x) \\ &\rightarrow_{\beta} \lambda fx.f^m (f^n x) \\ &= \lambda fx.f^{m+n} x \\ &= \overline{m + n} \end{aligned}$$

Am folosit compunerea lui  $f^m$  cu  $f^n$  pentru a obține  $f^{m+n}$ .

# Numere naturale

$$\bar{n} \triangleq \lambda fx.f^n x$$

$$\mathbf{Succ} \triangleq \lambda nfx.f (n f x)$$

Putem defini **adunarea** și ca aplicarea repetată a funcției succesor:

$$\mathbf{add}' \triangleq \lambda mn.m \mathbf{Succ} n$$

$$\begin{aligned}\mathbf{add}' \bar{m} \bar{n} &= (\lambda mn.m \mathbf{Succ} n) \bar{m} \bar{n} \\&\rightarrow_{\beta} \bar{m} \mathbf{Succ} \bar{n} \\&= (\lambda fx.f^m x) \mathbf{Succ} \bar{n} \\&\rightarrow_{\beta} \mathbf{Succ}^m \bar{n} \\&= \underbrace{\mathbf{Succ}(\mathbf{Succ}(\dots(\mathbf{Succ} \bar{n})\dots))}_m \\&\rightarrow_{\beta} \underbrace{\mathbf{Succ}(\mathbf{Succ}(\dots(\mathbf{Succ} \overline{n+1})\dots))}_{m-1} \\&\rightarrow_{\beta} \overline{m+n}\end{aligned}$$

$$\bar{n} \triangleq \lambda f x. f^n x \qquad \mathbf{Succ} \triangleq \lambda n f x. f (n f x)$$

$$\mathbf{add}' \triangleq \lambda m n. m \mathbf{Succ} n$$

Similar **înmulțirea** este adunare repetată, iar ridicarea la putere este înmulțire repetată:

$$\mathbf{mul} \triangleq \lambda m n. m (\mathbf{add} n) \bar{0}$$

$$\mathbf{exp} \triangleq \lambda m n. m (\mathbf{mul} n) \bar{1}$$

## Numere naturale

Putem defini o funcție de la numere naturale la booleani care verifică dacă un număr natural este 0 sau nu

$$\text{isZero}(0) = \text{true}$$

$$\text{isZero}(n) = \text{false} \quad \text{dacă } n \neq 0$$

O codare în lambda calcul a unei astfel de funcții este

$$\text{isZero} \triangleq \lambda nxy. n (\lambda z.y) x$$

**Exercițiu.** Verificați afirmația de mai sus.

Putem să definim și codarea **pred** pentru predecesorul unui număr natural. Această codare nu este deloc ușoară și alegem să lucrăm cu ea ca cu o cutie neagră.

## Putem exprima mai mult?

Avem văzut codari simple pentru booleeni și numere naturale.

Totuși nu avem o metodă pentru a construi astfel de  $\lambda$ -termeni.

Ne trebuie un mecanism care să ne permită să construim funcții mai complicate din funcții mai simple.

De exemplu, să considerăm funcția factorial

$$0! = 1$$

$$n! = n \cdot (n - 1)!, \quad \text{dacă } n \neq 0$$

## Puncte fixe

---



Fie  $f$  o funcție. Spunem că  $x$  este un **punct fix** al lui  $f$  dacă  $f(x) = x$ .

În matematică, unele funcții au puncte fixe, altele nu au.

De exemplu,  $f(x) = x^2$  are două puncte fixe 0 și 1, dar  $f(x) = x + 1$  nu are puncte fixe.

Mai mult, unele funcții au o infinitate de puncte fixe, cum ar fi  $f(x) = x$ .

## $\beta$ -echivalență

Am notat cu  $M \rightarrow_{\beta} M'$  faptul că  $M$  poate fi  $\beta$ -redus până la  $M'$  în 0 sau mai mulți pași de  $\beta$ -reducție.

$\rightarrow_{\beta}$  este închiderea reflexivă și tranzitivă a relației  $\rightarrow_{\beta}$ .

Notăm cu  $M =_{\beta} M'$  faptul că  $M$  poate fi transformat în  $M'$  în 0 sau mai mulți pași de  $\beta$ -reducție, transformare în care pașii de reducție pot fi și întorși.

$=_{\beta}$  este închiderea reflexivă, simetrică și tranzitivă a relației  $\rightarrow_{\beta}$ .

De exemplu, avem  $(\lambda y. y \ v) \ z =_{\beta} (\lambda x. z \ x) \ v$  deoarece avem

$$(\lambda y. y \ v) \ z \rightarrow_{\beta} z \ v \leftarrow_{\beta} (\lambda x. z \ x) \ v$$

Notăm cu  $\leftarrow_{\beta}$  inversul relației  $\rightarrow_{\beta}$ .

## Puncte fixe în lambda-calcul

Dacă  $F$  și  $M$  sunt  $\lambda$ -termeni, spunem că  $M$  este un **punct fix** al lui  $F$  dacă  $F M =_{\beta} M$ .

**Thm.** În lambda calculul fără tipuri, orice termen are un punct fix.

## Puncte fixe în lambda-calcul

Dacă  $F$  și  $M$  sunt  $\lambda$ -termeni, spunem că  $M$  este un **punct fix** al lui  $F$  dacă  $F M =_{\beta} M$ .

**Thm.** În lambda calculul fără tipuri, orice termen are un punct fix.

**Dem.** Vrem să arătăm că pentru orice termen  $F$  există un termen  $M$  astfel încât  $F M =_{\beta} M$ .

Fie  $F$  un termen. Considerăm  $M \triangleq (\lambda x. F (x x)) (\lambda x. F (x x))$ . Avem

$$\begin{aligned} M &= (\lambda x. F (x x)) (\lambda x. F (x x)) \\ &\rightarrow_{\beta} F ((\lambda x. F (x x)) (\lambda x. F (x x))) \\ &= F M \end{aligned}$$

Deci avem  $F M =_{\beta} M$ .

# Combinatori de punct fix

Combinatorii de puncte fixe sunt termeni închiși care "construiesc" un punct fix pentru un termen arbitrar.

Câteva exemple:

- Combinatorul de punct fix al lui Curry

$$\mathbf{Y} \triangleq \lambda y. (\lambda x. y (x x)) (\lambda x. y (x x))$$

Pentru orice termen  $F$ ,  $\mathbf{Y}F$  este un punct fix al lui  $F$  deoarece  $\mathbf{Y}F \rightarrow_{\beta} F (\mathbf{Y}F)$ .

- Combinatorul de punct fix al lui Turing

$$\Theta \triangleq (\lambda x y. y (x x y)) (\lambda x y. y (x x y))$$

Pentru orice termen  $F$ ,  $\Theta F$  este un punct fix al lui  $F$  deoarece  $\Theta F \rightarrow_{\beta} F (\Theta F)$ .

## Rezolvarea de ecuații în lambda calcul

Punctele fixe ne permit să rezolvăm ecuații. A găsi un punct fix pentru  $f$  este același lucru cu a rezolva o ecuație de forma

$$x = f(x)$$

Am văzut că în lambda calcul există mereu o soluție pentru astfel de ecuații.

## Rezolvarea de ecuații în lambda calcul

Să aplicăm această idee pentru funcția factorial.

Cea mai naturală definiție a funcției factorial este cea recursivă și o putem scrie în lambda calcul prin

$$\mathbf{fact} \, n = \mathbf{if} \, (\mathbf{isZero} \, n) \, (\overline{1}) \, (\mathbf{mul} \, n \, (\mathbf{fact}(\mathbf{pred} \, n)))$$

În ecuația de mai sus, **fact** apare și în stânga, și în dreapta. Pentru a găsi cine este **fact**, trebuie să rezolvăm o ecuație.

## Rezolvarea de ecuații în lambda calcul

Să rezolvăm ecuația de mai sus. Rescriem problema puțin

$$\begin{aligned}\mathbf{fact} &= \lambda n. \mathbf{if} (\mathbf{isZero} \ n) (\bar{1}) (\mathbf{mul} \ n (\mathbf{fact}(\mathbf{pred} \ n))) \\ \mathbf{fact} &= (\lambda fn. \mathbf{if} (\mathbf{isZero} \ n) (\bar{1}) (\mathbf{mul} \ n (f(\mathbf{pred} \ n))) ) \mathbf{fact}\end{aligned}$$

Notăm termenul  $\lambda fn. \mathbf{if} (\mathbf{isZero} \ n) (\bar{1}) (\mathbf{mul} \ n (f(\mathbf{pred} \ n)))$  cu  $F$ .

Ultima ecuație devine  $\mathbf{fact} = F \mathbf{fact}$ , o ecuație de punct fix.

Am văzut că  $\mathbf{Y} F$  este un punct fix pentru  $F$  (adică  $\mathbf{Y} F \rightarrow_{\beta} F (\mathbf{Y} F)$ ), de aceea putem rezolva ecuația de mai sus luând

$$\begin{aligned}\mathbf{fact} &\triangleq \mathbf{Y} F \\ \mathbf{fact} &\triangleq \mathbf{Y} (\lambda fn. \mathbf{if} (\mathbf{isZero} \ n) (\bar{1}) (\mathbf{mul} \ n (f(\mathbf{pred} \ n))) )\end{aligned}$$

Observați că  $\mathbf{fact}$  a dispărut din partea dreaptă.

**Exercițiu.** Evaluați  $\mathbf{fact} \bar{2}$  ținând cont că  $\mathbf{fact} \rightarrow_{\beta} F \mathbf{fact}$ .



Quiz time!



<https://tinyurl.com/C04-Quiz1>

**Pe săptămâna viitoare!**

# Fundamentele limbajelor de programare

C05

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## **Lambda calcul cu tipuri simple**

---

# Probleme cu lambda calculul fără tipuri

## Proprietăți negative ale lambda calculului fără tipuri:

- Aplicații de forma  $x x$  sau  $M M$  sunt permise, deși sunt contraintuitive.
- Existența formelor normale pentru  $\lambda$ -termeni nu este garantată și putem avea "calcul infinite" nedorite
- Orice  $\lambda$ -termen are un punct fix ceea ce nu este în armonie cu ceea ce știam despre funcții oarecare

Vrem să eliminăm aceste proprietăți negative, păstrându-le pe cele pozitive.

Proprietățile negative sunt eliminate prin adăugarea de **tipuri** ceea ce induce restricțiile dorite pe termeni.

# Tipuri simple

Fie  $\mathbb{V} = \{\alpha, \beta, \gamma, \dots\}$  o mulțime infinită de **tipuri variabilă**.

Mulțimea tuturor **tipurilor simple**  $\mathbb{T}$  este definită prin

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \rightarrow \mathbb{T}$$

- (**Tipul variabilă**) Dacă  $\alpha \in \mathbb{V}$ , atunci  $\alpha \in \mathbb{T}$ .
- (**Tipul săgeată**) Dacă  $\sigma, \tau \in \mathbb{T}$ , atunci  $(\sigma \rightarrow \tau) \in \mathbb{T}$ .

Câteodată vom nota tipurile simple și cu litere  $A, B, \dots$

Tipurile variabilă sunt reprezentări abstracte pentru **tipuri de bază** cum ar fi *Nat* pentru numere naturale, *List* pentru liste etc.

Tipurile săgeată reprezintă **tipuri pentru funcții** cum ar fi

- $Nat \rightarrow Real$ , mulțimea tuturor funcțiilor de la numere naturale la numere reale
- $(Nat \rightarrow Int) \rightarrow (Int \rightarrow Nat)$ , mulțimea tuturor funcțiilor care au ca intrare o funcție de la numere naturale la întregi și produce o funcție de la întregi la numere naturale.

# Tipuri simple

Mulțimea tipurilor simple  $T = V \mid T \rightarrow T$

Exemple de tipuri simple:

- $\gamma$
- $(\beta \rightarrow \gamma)$
- $((\gamma \rightarrow \alpha) \rightarrow (\alpha \rightarrow (\beta \rightarrow \gamma)))$

În tipurile săgeată, parantezele exterioare pot fi omise.

Parantezele în tipurile săgeată sunt asociative la dreapta.

De exemplu,

- $\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3 \rightarrow \alpha_4$  este abreviere pentru  $(\alpha_1 \rightarrow (\alpha_2 \rightarrow (\alpha_3 \rightarrow \alpha_4)))$
- $x_1 \ x_2 \ x_3 \ x_4$  este abreviere pentru  $((((x_1 \ x_2) \ x_3) \ x_4)$

## Termeni și tipuri

Ce înseamnă că un termen  $M$  are un tip  $\sigma$ ?

Vom nota acest lucru cu  $M:\sigma$ .



## Termeni și tipuri

Ce înseamnă că un termen  $M$  are un tip  $\sigma$ ?

Vom nota acest lucru cu  $M : \sigma$ .

**Variabilă.** Dacă o variabilă  $x$  are un tip  $\sigma$ , notăm cu  $x : \sigma$ .

**Convenția Barendregt:** variabilele legate sunt distincte.

Presupunem că orice variabilă din  $M$  are un unic tip.

Dacă  $x : \sigma$  și  $x : \tau$ , atunci  $\sigma \equiv \tau$ .

## Termeni și tipuri

Ce înseamnă că un termen  $M$  are un tip  $\sigma$ ?

Vom nota acest lucru cu  $M : \sigma$ .

**Variabilă.** Dacă o variabilă  $x$  are un tip  $\sigma$ , notăm cu  $x : \sigma$ .

**Convenția Barendregt:** variabilele legate sunt distincte.

Presupunem că orice variabilă din  $M$  are un unic tip.

Dacă  $x : \sigma$  și  $x : \tau$ , atunci  $\sigma \equiv \tau$ .

**Aplicare.** Pentru  $M N$  este clar că vrem să știm tipurile lui  $M$  și  $N$ . Intuitiv,  $M N$  înseamnă că ("funcția")  $M$  este aplicată ("intrării")  $N$ . Atunci  $M$  trebuie să aibă un tip funcție, adică  $M : \sigma \rightarrow \tau$ , iar  $N$  trebuie să fie "adecvat" pentru această funcție, adică  $N : \sigma$ .

Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

## Termeni și tipuri

Ce înseamnă că un termen  $M$  are un tip  $\sigma$ ?

Vom nota acest lucru cu  $M : \sigma$ .

**Variabilă.** Dacă o variabilă  $x$  are un tip  $\sigma$ , notăm cu  $x : \sigma$ .

**Convenția Barendregt:** variabilele legate sunt distincte.

Presupunem că **orice variabilă din  $M$  are un unic tip**.

Dacă  $x : \sigma$  și  $x : \tau$ , atunci  $\sigma \equiv \tau$ .

**Aplicare.** Pentru  $M N$  este clar că vrem să știm tipurile lui  $M$  și  $N$ .

Intuitiv,  $M N$  înseamnă că ("funcția")  $M$  este aplicată ("intrării")  $N$ .

Atunci  $M$  trebuie să aibă un tip funcție, adică  $M : \sigma \rightarrow \tau$ , iar  $N$  trebuie să fie "adecvat" pentru această funcție, adică  $N : \sigma$ .

Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

**Abstractizare.** Dacă  $M : \tau$ , ce tip trebuie să aibă  $\lambda x. M$ ?

Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

## Termeni și tipuri

Variabilă.  $x : \sigma$ .

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

$M$  *are tip* (este *typeable*) dacă există un tip  $\sigma$  astfel încât  $M : \sigma$ .

# Termeni și tipuri

Variabilă.  $x : \sigma$ .

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

$M$  are tip (este *typeable*) dacă există un tip  $\sigma$  astfel încât  $M : \sigma$ .

Exemple.

- Dacă  $x : \sigma$ , atunci funcția identitate are tipul  $\lambda x. x : \sigma \rightarrow \sigma$ .

# Termeni și tipuri

Variabilă.  $x : \sigma$ .

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

$M$  are tip (este *typeable*) dacă există un tip  $\sigma$  astfel încât  $M : \sigma$ .

Exemple.

- Dacă  $x : \sigma$ , atunci funcția identitate are tipul  $\lambda x. x : \sigma \rightarrow \sigma$ .
- Conform convențiilor de la aplicare,  $y x$  poate avea un tip doar dacă  $y$  are un tip săgeată de forma  $\sigma \rightarrow \tau$  și tipul lui  $x$  se potrivește cu tipul domeniu  $\sigma$ . În acest caz, tipul lui  $y x : \tau$ .

# Termeni și tipuri

Variabilă.  $x : \sigma$ .

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

$M$  are tip (este *typeable*) dacă există un tip  $\sigma$  astfel încât  $M : \sigma$ .

Exemple.

- Dacă  $x : \sigma$ , atunci funcția identitate are tipul  $\lambda x. x : \sigma \rightarrow \sigma$ .
- Conform convențiilor de la aplicare,  $y x$  poate avea un tip doar dacă  $y$  are un tip săgeată de forma  $\sigma \rightarrow \tau$  și tipul lui  $x$  se potrivește cu tipul domeniu  $\sigma$ . În acest caz, tipul lui  $y x : \tau$ .
- **Termenul  $x x$  nu poate avea nici un tip** (nu este typeable).  
Pe de o parte,  $x$  ar trebui să aibă tipul  $\sigma \rightarrow \tau$  (pentru prima apariție), pe de altă ar trebui să aibă tipul  $\sigma$  (pentru a doua apariție). Cum am stabilit că orice variabilă are un unic tip, obținem  $\sigma \rightarrow \tau \equiv \sigma$ , ceea ce este imposibil.

## Discuție despre asociativitate

Asociativitatea la dreapta pentru tipurile săgeată vs. asociativitatea la stânga pentru aplicare:

- Să presupunem că  $f:\rho \rightarrow (\sigma \rightarrow \tau)$ ,  $x:\rho$  și  $y:\sigma$ .
- Atunci  $f\ x:\sigma \rightarrow \tau$  și  $(f\ x)\ y:\tau$ .



## Discuție despre asociativitate

Asociativitatea la dreapta pentru tipurile săgeată vs. asociativitatea la stânga pentru aplicare:

- Să presupunem că  $f:\rho \rightarrow (\sigma \rightarrow \tau)$ ,  $x:\rho$  și  $y:\sigma$ .
- Atunci  $f\ x:\sigma \rightarrow \tau$  și  $(f\ x)\ y:\tau$ .
- Folosind ambele convenții pentru asociativitate pentru a elimina parantezele, avem

$$f:\rho \rightarrow \sigma \rightarrow \tau$$

$$f\ x\ y:\tau$$

Convențiile pentru asociativitate sunt în armonie una cu cealaltă.

## Church-typing vs. Curry-typing

A găsi tipul unui termen începe cu a găsi tipurile pentru variabile.  
Există două metode prin care putem asocia tipuri variabilelor.

### Asociere explicită (*Church-typing*).

- Constă în prescrierea unui unic tip pentru fiecare variabilă, la introducerea acesteia.
- Presupune că tipurile variabilelor sunt explicit stabilite.
- Tipurile termenilor mai complecși se obțin natural, ținând cont de convențiile pentru aplicare și abstractizare.

### Asociere implicită (*Curry-typing*).

- Constă în a nu prescrie un tip pentru fiecare variabilă, ci în a le lăsa "deschise" (implicite).
- În acest caz, termenii *typeable* sunt descoperiți printr-un proces de căutare, care poate presupune "ghicirea" anumitor tipuri.

# Church-typing vs. Curry-typing

Exemplu. Asociere explicită (*Church-typing*).

Vrem să calculăm tipul expresiei  $(\lambda zu. z) (y x)$  știind că

1.  $x : \alpha \rightarrow \alpha$

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ ,

2.  $y : (\alpha \rightarrow \alpha) \rightarrow \beta$

atunci  $M N : \tau$ .

3.  $z : \beta$

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ ,

4.  $u : \gamma$

atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

# Church-typing vs. Curry-typing

Exemplu. Asociere explicită (*Church-typing*).

Vrem să calculăm tipul expresiei  $(\lambda z u. z) (y x)$  știind că

- |  |   |
|--|---|
| 1. $x : \alpha \rightarrow \alpha$                     | <b>Aplicare.</b> Dacă $M : \sigma \rightarrow \tau$ și $N : \sigma$ , |
| 2. $y : (\alpha \rightarrow \alpha) \rightarrow \beta$ | atunci $M N : \tau$ .   |
| 3. $z : \beta$   | <b>Abstractizare.</b> Dacă $x : \sigma$ și $M : \tau$ ,               |
| 4. $u : \gamma$  | atunci $\lambda x. M : \sigma \rightarrow \tau$ .                     |

Din (2) și (1), prin aplicare obținem (5):  $y x : \beta$ .

Din (4) și (3), prin abstractizare obținem (6):  $\lambda u. z : \gamma \rightarrow \beta$ .

Din (3) și (6), prin abstractizare obținem (7):  $\lambda z u. z : \beta \rightarrow \gamma \rightarrow \beta$ .

Nu uitați că  $\beta \rightarrow \gamma \rightarrow \beta$  înseamnă  $\beta \rightarrow (\gamma \rightarrow \beta)$ .

Atunci, din (7) și (5), prin aplicare, avem  $(\lambda z u. z) (y x) : \gamma \rightarrow \beta$ .

# Church-typing vs. Curry-typing

Exemplu. Asociere implicită (Curry-typing).

Considerăm termenul de mai devreme  $M = (\lambda z u. z) (y x)$ .

Putem să "ghicim" tipurile variabilelor astfel încât  $M$  să aibă tip?

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

# Church-typing vs. Curry-typing

Exemplu. Asociere implicită (Curry-typing).

Considerăm termenul de mai devreme  $M = (\lambda z u. z) (y x)$ .

Putem să "ghicim" tipurile variabilelor astfel încât  $M$  să aibă tip?

Aplicare. Dacă  $M : \sigma \rightarrow \tau$  și  $N : \sigma$ , atunci  $M N : \tau$ .

Abstractizare. Dacă  $x : \sigma$  și  $M : \tau$ , atunci  $\lambda x. M : \sigma \rightarrow \tau$ .

- Observăm că  $M$  este o aplicare a lui  $\lambda z u. z$  termenului  $y x$ .
- Atunci  $\lambda z u. z$  trebuie să aibă un tip săgeată, de exemplu  $\lambda z u. z : A \rightarrow B$ , și  $y x$  să se potrivească, adică  $y x : A$ .
- În acest caz, avem  $M : B$ .

# Church-typing vs. Curry-typing

Exemplu. Asociere implicită (*Curry-typing*) (cont.)

Știm  $M = (\lambda z u. z) (y x)$  și am dedus până acum:

$$\lambda z u. z : A \rightarrow B \quad y x : A \quad M : B$$

- Faptul că  $\lambda z u. z : A \rightarrow B$  implică că  $z : A$  și  $\lambda u. z : B$ .
- Deducem că  $B$  este tipul unei abstractizări, deci  $B \equiv C \rightarrow D$ , și obținem că  $u : C$  și  $z : D$ .
- Pe de altă parte,  $y x$  este o aplicare, deci trebuie să existe  $E$  și  $F$  astfel încât  $y : E \rightarrow F$  și  $x : E$ . Atunci  $y x : F$ .

## Church-typing vs. Curry-typing

Exemplu. Asociere implicită (Curry-typing) (cont.)

Știm  $M = (\lambda zu. z) (y x)$ . Am dedus următoarele:

- $x : E$
- $y : E \rightarrow F$
- $z : A$  și  $z : D$ , deci  $A \equiv D$
- $u : C$
- $B \equiv C \rightarrow D$
- $y x : A$  și  $y x : F$ , deci  $A \equiv F$ .

În concluzie,  $A \equiv D \equiv F$ , și eliminând redundanțele obținem

$$(*) \quad x : E \quad y : E \rightarrow A \quad z : A \quad u : C$$

Reamintim că aveam  $M : B$ , adică  $M : C \rightarrow A$ .

Am obținut o schemă generală (\*) pentru tipurile lui  $x, y, z, u$  care induc un tip pentru  $M$ .



# Church-typing vs. Curry-typing

Exemplu. Asociere implicită (*Curry-typing*) (cont.)

Știm  $M = (\lambda zu. z) (y x)$ . Am obținut schema generală

$$(*) \quad x:E \quad y:E \rightarrow A \quad z:A \quad u:C \quad M:C \rightarrow A$$

În schema de mai sus, putem considera tipuri "reale":

- $x:\beta, \quad y:\beta \rightarrow \alpha, \quad z:\alpha, \quad u:\delta, \quad M:\delta \rightarrow \alpha$

# Church-typing vs. Curry-typing

Exemplu. Asociere implicită (*Curry-typing*) (cont.)

Știm  $M = (\lambda zu. z) (y x)$ . Am obținut schema generală

$$(*) \quad x:E \quad y:E \rightarrow A \quad z:A \quad u:C \quad M:C \rightarrow A$$

În schema de mai sus, putem considera tipuri "reale":

- $x:\beta, \quad y:\beta \rightarrow \alpha, \quad z:\alpha, \quad u:\delta, \quad M:\delta \rightarrow \alpha$
- $x:\alpha \rightarrow \alpha, \quad y:(\alpha \rightarrow \alpha) \rightarrow \beta, \quad z:\beta, \quad u:\gamma, \quad M:\gamma \rightarrow \beta$   
(soluția discutată la Church-typing)

# Church-typing vs. Curry-typing

Exemplu. Asociere implicită (Curry-typing) (cont.)

Știm  $M = (\lambda zu. z) (y x)$ . Am obținut schema generală

$$(*) \quad x:E \quad y:E \rightarrow A \quad z:A \quad u:C \quad M:C \rightarrow A$$

În schema de mai sus, putem considera tipuri "reale":

- $x:\beta, \quad y:\beta \rightarrow \alpha, \quad z:\alpha, \quad u:\delta, \quad M:\delta \rightarrow \alpha$
- $x:\alpha \rightarrow \alpha, \quad y:(\alpha \rightarrow \alpha) \rightarrow \beta, \quad z:\beta, \quad u:\gamma, \quad M:\gamma \rightarrow \beta$   
(soluția discutată la Church-typing)
- $x:\alpha, \quad y:\alpha \rightarrow \alpha \rightarrow \beta, \quad z:\alpha \rightarrow \beta, \quad u:\alpha \rightarrow \alpha, \quad M:(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta$

## Church-typing vs. Curry-typing

Asocierea implicită de tipuri (*Curry-typing*) are proprietăți interesante, cum am văzut în exemplul anterior.

Totuși, în continuare vom folosi asocierea explicită (*Church-typing*) deoarece de obicei tipurile sunt cunoscute dinainte (și declararea tipurilor pentru argumentele unei funcții este o bună-practică).

Marcăm tipurile **variabilelor legate** imediat după introducerea lor cu o abstractizare. Tipurile **variabilelor libere** sunt date de un **context**.

**Exemplu.** Să considerăm exemplul anterior  $(\lambda z u. z) (y x)$ .

Observați că  $z$  și  $u$  sunt legate, iar  $x$  și  $y$  sunt libere.

Presupunând că  $z:\beta$  și  $u:\gamma$ , scriem termenul astfel

$$(\lambda z:\beta. \lambda u:\gamma. z) (y x)$$

## Church-typing

**Exemplu.** Să considerăm exemplul anterior  $(\lambda z u. z) (y x)$ .

Observați că  $z$  și  $u$  sunt legate, iar  $x$  și  $y$  sunt libere.

Presupunând că  $z:\beta$  și  $u:\gamma$ , scriem termenul astfel

$$(\lambda z:\beta. \lambda u:\gamma. z) (y x)$$

Dacă presupunem un context în care despre variabilele libere știm, de exemplu, că  $x:\alpha \rightarrow \alpha$  și  $y:(\alpha \rightarrow \alpha) \rightarrow \beta$ , atunci folosim notația:

$$x:\alpha \rightarrow \alpha, y:(\alpha \rightarrow \alpha) \rightarrow \beta \vdash (\lambda z:\beta. \lambda u:\gamma. z) (y x)$$

**Exemplu.** Să considerăm exemplul anterior  $(\lambda z u. z) (y x)$ .

Observați că  $z$  și  $u$  sunt legate, iar  $x$  și  $y$  sunt libere.

Presupunând că  $z:\beta$  și  $u:\gamma$ , scriem termenul astfel

$$(\lambda z:\beta. \lambda u:\gamma. z) (y x)$$

Dacă presupunem un context în care despre variabilele libere știm, de exemplu, că  $x:\alpha \rightarrow \alpha$  și  $y:(\alpha \rightarrow \alpha) \rightarrow \beta$ , atunci folosim notația:

$$x:\alpha \rightarrow \alpha, y:(\alpha \rightarrow \alpha) \rightarrow \beta \vdash (\lambda z:\beta. \lambda u:\gamma. z) (y x)$$

Încă nu avem o noțiune de  $\beta$ -reducție pentru termeni cu tipuri, dar ne-am putea gândi că am avea:

$$(\lambda z:\beta. \lambda u:\gamma. z) (y x) \rightarrow_{\beta} \lambda u:\gamma. y x.$$

Observați că am dori să deducem că  $(\lambda u:\gamma. y x):\gamma \rightarrow \beta$ .

## Sistem de deducție pentru Church $\lambda \rightarrow$

Deoarece am convenit cum să decorăm cu informații despre tipuri variabilele legate, trebuie să actualizăm definiția  $\lambda$ -termenilor.

Mulțimea  $\lambda$ -termenilor cu pre-tipuri  $\Lambda_T$  este

$$\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T. \Lambda_T$$

O **afirmație** este o expresie de forma  $M : \sigma$ , unde  $M \in \Lambda_T$  și  $\sigma \in T$ .

Într-o astfel de afirmație,  $M$  se numește **subiect** și  $\sigma$  **tip**.

O **declarație** este o afirmație în care subiectul este o variabilă ( $x : \sigma$ ).

Un **context** este o listă de declarații cu subiecți diferiți.

O **judcată** este o expresie de forma  $\Gamma \vdash M : \sigma$ , unde  $\Gamma$  este context și  $M : \sigma$  este o afirmație.



## Sistem de deducție pentru Church $\lambda \rightarrow$

Deoarece suntem în general interesați de termeni *typeable*, am dori să avem o metodă prin care să putem stabili dacă un termen  $t \in \Lambda_{\mathbb{T}}$  este *typeable* și dacă da, să calculăm un tip pentru  $t$ .

Vom da niște reguli care să ne permită să stabilim dacă o judecată  $\Gamma \vdash M : \sigma$  poate fi dedusă, adică dacă  $M$  are tipul  $\sigma$  în contextul  $\Gamma$ .

## Sistem de deducție pentru calculul Church $\lambda \rightarrow$

$$\frac{}{\Gamma \vdash x:\sigma} \text{dacă } x:\sigma \in \Gamma \text{ (var)}$$

$$\frac{\Gamma \vdash M:\sigma \rightarrow \tau \quad \Gamma \vdash N:\sigma}{\Gamma \vdash MN:\tau} \text{ (app)}$$

$$\frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash (\lambda x:\sigma. M):\sigma \rightarrow \tau} \text{ (abs)}$$

Un termen  $M$  în calculul  $\lambda \rightarrow$  este **legal** dacă există un context  $\Gamma$  și un tip  $\rho$  astfel încât  $\Gamma \vdash M:\rho$ .

## Sistem de deducție pentru calculul Church $\lambda \rightarrow$

$$\frac{}{\Gamma \vdash x:\sigma} \text{ (var)} \quad \frac{\Gamma \vdash M:\sigma \rightarrow \tau \quad \Gamma \vdash N:\sigma}{\Gamma \vdash MN:\tau} \text{ (app)} \quad \frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash (\lambda x:\sigma. M):\sigma \rightarrow \tau} \text{ (abs)}$$

dacă  $x:\sigma \in \Gamma$

**Exemplu.** Să arătăm că termenul  $\lambda y:\alpha \rightarrow \beta. \lambda z:\alpha. yz$  are tipul  $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$  în contextul vid.

$$\emptyset \vdash (\lambda y:\alpha \rightarrow \beta. \lambda z:\alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$$

# Sistem de deducție pentru calculul Church $\lambda \rightarrow$

$$\frac{}{\Gamma \vdash x:\sigma} \text{ (var)} \quad \frac{\Gamma \vdash M:\sigma \rightarrow \tau \quad \Gamma \vdash N:\sigma}{\Gamma \vdash MN:\tau} \text{ (app)} \quad \frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash (\lambda x:\sigma. M):\sigma \rightarrow \tau} \text{ (abs)}$$

dacă  $x:\sigma \in \Gamma$

**Exemplu.** Să arătăm că termenul  $\lambda y:\alpha \rightarrow \beta. \lambda z:\alpha. yz$  are tipul  $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$  în contextul vid.

$$\emptyset \vdash (\lambda y:\alpha \rightarrow \beta. \lambda z:\alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$$

$$\frac{\frac{\frac{}{y:\alpha \rightarrow \beta, z:\alpha \vdash y:\alpha \rightarrow \beta} \text{ (var)} \quad \frac{}{y:\alpha \rightarrow \beta, z:\alpha \vdash z:\alpha} \text{ (var)}}{y:\alpha \rightarrow \beta, z:\alpha \vdash (yz):\beta} \text{ (app)}}{y:\alpha \rightarrow \beta \vdash (\lambda z:\alpha. yz):\alpha \rightarrow \beta} \text{ (abs)} \quad \frac{}{\emptyset \vdash (\lambda y:\alpha \rightarrow \beta. \lambda z:\alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta} \text{ (abs)}$$

## Diferite stiluri pentru a scrie deducții

În exemplul anterior, am scris derivarea în **stilul arbore**.

În **stilul liniar**, derivarea precedentă ar arăta astfel:

1.  $y : \alpha \rightarrow \beta, z : \alpha \vdash y : \alpha \rightarrow \beta$  (var)
2.  $y : \alpha \rightarrow \beta, z : \alpha \vdash z : \alpha$  (var)
3.  $y : \alpha \rightarrow \beta, z : \alpha \vdash (yz) : \beta$  (app) cu 1 și 2
4.  $y : \alpha \rightarrow \beta \vdash (\lambda z : \alpha. yz) : \alpha \rightarrow \beta$  (abs) cu 3
5.  $\emptyset \vdash (\lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$  (abs) cu 4

## Diferite stiluri pentru a scrie deducții

În **stilul cu cutii**, afișăm fiecare declarație la începutul unei cutii și considerăm că declarația respectivă face parte din contextul pentru toate afirmațiile din cutia respectivă.

Când închidem o cutie, abstractizăm după variabila din declarația de la începutul cutiei.

$$(\lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$$

## Diferite stiluri pentru a scrie deducții

În **stilul cu cutii**, afișăm fiecare declarație la începutul unei cutii și considerăm că declarația respectivă face parte din contextul pentru toate afirmațiile din cutia respectivă.

Când închidem o cutie, abstractizăm după variabila din declarația de la începutul cutiei.

$$\frac{\begin{array}{l} y : \alpha \rightarrow \beta \quad (\text{context}) \\ (\lambda z : \alpha. yz) : \alpha \rightarrow \beta \end{array}}{(\lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta \quad (\text{abs})}$$

## Diferite stiluri pentru a scrie deducții

În **stilul cu cutii**, afișăm fiecare declarație la începutul unei cutii și considerăm că declarația respectivă face parte din contextul pentru toate afirmațiile din cutia respectivă.

Când închidem o cutie, abstractizăm după variabila din declarația de la începutul cutiei.

$$\frac{\frac{\frac{y : \alpha \rightarrow \beta}{(context)} \quad \frac{z : \alpha \quad (context)}{(yz) : \beta}}{(\lambda z : \alpha. yz) : \alpha \rightarrow \beta \quad (abs)}}{(\lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta \quad (abs)}$$



## Diferite stiluri pentru a scrie deducții

În **stilul cu cutii**, afișăm fiecare declarație la începutul unei cutii și considerăm că declarația respectivă face parte din contextul pentru toate afirmațiile din cutia respectivă.

Când închidem o cutie, abstractizăm după variabila din declarația de la începutul cutiei.

- |    |  |                 |
|----|--|-----------------|
| 1. | $y : \alpha \rightarrow \beta$   | (context)       |
| 2. | $z : \alpha$   | (context)       |
| 3. | $(yz) : \beta$   | (app) cu 1 și 2 |
| 4. | $(\lambda z : \alpha. yz) : \alpha \rightarrow \beta$  | (abs) cu 3      |
| 5. | $(\lambda y : \alpha \rightarrow \beta. \lambda z : \alpha. yz) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$ | (abs) cu 4      |

**Exercițiu.** Arătați că termenul  $\lambda x : ((\alpha \rightarrow \beta) \rightarrow \alpha). x (\lambda z : \alpha. y)$  are tipul  $((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$  în contextul  $y : \beta$ .

Quiz time!



<https://tinyurl.com/2p9xf67e>

**Pe săptămâna viitoare!**

# Fundamentele limbajelor de programare

C06

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## **Lambda calcul cu tipuri simple (cont.)**

---

# Tipuri simple

Mulțimea **tipurilor simple**  $T = V \mid T \rightarrow T$

- (Tipul variabilă) Dacă  $\alpha \in V$ , atunci  $\alpha \in T$ .
- (Tipul săgeată) Dacă  $\sigma, \tau \in T$ , atunci  $(\sigma \rightarrow \tau) \in T$ .

Mulțimea  **$\lambda$ -termenilor cu pre-tipuri**  $\Lambda_T \quad \Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T. \Lambda_T$

- O **afirmație** este o expresie de forma  $M : \sigma$ , unde  $M \in \Lambda_T$  și  $\sigma \in T$ .
- O **declarație** este o afirmație de forma  $x : \sigma$ .
- Un **context**  $\Gamma$  este o listă de declarații cu subiecți diferiți.
- O **judecată** este o expresie de forma  $\Gamma \vdash M : \sigma$ .

**Sistem de deducție pentru calculul Church  $\lambda \rightarrow$**

$$\frac{}{\Gamma \vdash x : \sigma} \text{ (var)} \quad \frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E) \quad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

dacă  $x : \sigma \in \Gamma$

Un termen  $M$  în calculul  $\lambda \rightarrow$  este **legal** dacă  $\Gamma \vdash M : \rho$ .

# Ce probleme putem să rezolvăm în teoria tipurilor?

## *Type Checking*

Se reduce la a verifica că putem găsi o derivare pentru

$$\text{context} \vdash \text{term} : \text{type}$$



## Ce probleme putem să rezolvăm în teoria tipurilor?

### *Well-typedness (Typability)*

Se reduce la a verifica dacă un termen este **legal**. Concret, trebuie să găsim un context și un tip dacă termenul este legal, altfel să arătăm de ce nu se poate.

$$? \vdash \text{term} : ?$$

O variațiune a problemei este *Type Assignment* în care contextul este dat și trebuie să găsim tipul.

$$\text{context} \vdash \text{term} : ?$$

## Ce probleme putem să rezolvăm în teoria tipurilor?

### *Term Finding (Inhabitation)*

Dându-se un context și un tip, să stabilim dacă există un termen cu acel tip, în contextul dat.

$$\text{context} \vdash ? : \text{type}$$

Toate aceste probleme sunt decidabile pentru calculul Church  $\lambda \rightarrow$ !

## Limitări ale lambda-calculului cu tipuri simple

Nu mai avem recursie nelimitată deoarece combinatorii de punct fix nu sunt *typeable*.

De exemplu,  $Y \triangleq \lambda y. (\lambda x. y (x x)) (\lambda x. y (x x))$  nu este typeable.

Dar avem recursie primitivă (recursie care permite doar *looping* în care numărul de iterații este cunoscut dinainte).

De exemplu,  $\text{add} \triangleq \lambda m n f x. m f (n f x)$  este o funcție primitiv recursivă.

Faptul că orice evaluare se termină este important pentru implementări ale logicilor folosind lambda-calculul.

# Limitări ale lambda-calculului cu tipuri simple

Tipurile pot fi prea restrictive.

De exemplu, am putea gândi că termenul  $(\lambda f. \text{if } (f \mathbf{T}) (f 3) (f 5)) (\lambda x. x)$  ar trebui să aibă un tip. Dar nu are!

Soluții posibile:

- **Let-polymorphism** unde variabilele libere din tipul lui  $f$  se redenumesc la fiecare folosire. De exemplu, am putea scrie

$$\text{let } f = \lambda x. x \text{ in} \\ \text{if } (f \mathbf{T}) (f 3) (f 5)$$

- **Cuantificatori de tipuri.** De exemplu, am avea

$$\lambda x. x : \Pi \alpha . \alpha \rightarrow \alpha$$

Operatorul de legare  $\Pi$  face explicit faptul că variabila de tip  $\alpha$  nu este rigidă.

## Alte tipuri

---

## Tipul Unit și constructorul unit

Mulțimea tipurilor

$$T = V \mid T \rightarrow T \mid \text{Unit}$$

Mulțimea  $\lambda$ -termenilor cu pre-tipuri  $\Lambda_T$

$$\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x:T. \Lambda_T \mid \text{unit}$$

$$\frac{}{\Gamma \vdash \text{unit} : \text{Unit}} \text{ (unit)}$$

Mulțimea **tipurilor**

$$T = V \mid T \rightarrow T \mid \text{Unit} \mid \text{Void}$$

Mulțimea  **$\lambda$ -termenilor** cu pre-tipuri  $\Lambda_T$

$$\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x:T. \Lambda_T \mid \text{unit}$$

Nu există regulă de tipuri pentru deoarece tipul **Void** nu are inhabitant.

# Tipul produs și constructorul pereche

Mulțimea **tipurilor**

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \rightarrow \mathbb{T} \mid \text{Unit} \mid \text{Void} \mid \mathbb{T} \times \mathbb{T}$$

Mulțimea  **$\lambda$ -termenilor cu pre-tipuri**  $\Lambda_{\mathbb{T}}$

$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}} \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}. \Lambda_{\mathbb{T}} \mid \text{unit} \mid \langle \Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}} \rangle$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau} (\times_I)$$



# Tipul produs și constructorul pereche

Mulțimea **tipurilor**

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \rightarrow \mathbb{T} \mid \text{Unit} \mid \text{Void} \mid \mathbb{T} \times \mathbb{T}$$

Mulțimea  **$\lambda$ -termenilor cu pre-tipuri**  $\Lambda_{\mathbb{T}}$

$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}} \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}. \Lambda_{\mathbb{T}} \mid \text{unit} \mid \langle \Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}} \rangle \mid \text{fst } \Lambda_{\mathbb{T}} \mid \text{snd } \Lambda_{\mathbb{T}}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau} (\times_I)$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{fst } M : \sigma} (\times_{E_1}) \quad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{snd } M : \tau} (\times_{E_2})$$

# Tipul sumă și constructorii Left/Right

Mulțimea **tipurilor**

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \rightarrow \mathbb{T} \mid \text{Unit} \mid \text{Void} \mid \mathbb{T} \times \mathbb{T} \mid \mathbb{T} + \mathbb{T}$$

Mulțimea  **$\lambda$ -termenilor** cu pre-tipuri  $\Lambda_{\mathbb{T}}$

$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}} \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}. \Lambda_{\mathbb{T}} \mid \text{unit} \mid \langle \Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}} \rangle \mid \text{fst } \Lambda_{\mathbb{T}} \mid \text{snd } \Lambda_{\mathbb{T}} \\ \mid \text{Left } \Lambda_{\mathbb{T}} \mid \text{Right } \Lambda_{\mathbb{T}} \mid \text{case } \Lambda_{\mathbb{T}} \text{ of } \Lambda_{\mathbb{T}} ; \Lambda_{\mathbb{T}}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \text{Left } M : \sigma + \tau} (+_{l_1}) \quad \frac{\Gamma \vdash M : \tau}{\Gamma \vdash \text{Right } M : \sigma + \tau} (+_{l_2})$$

$$\frac{\Gamma \vdash M : \sigma + \tau \quad \Gamma \vdash M_1 : \sigma \rightarrow \gamma \quad \Gamma \vdash M_2 : \tau \rightarrow \gamma}{\Gamma \vdash \text{case } M \text{ of } M_1 ; M_2 : \gamma} (+_E)$$

# **Corespondența Curry-Howard**

---

# Schimbați perspectiva



Roger Antonsen  
Universitatea din Oslo

TED Talk: Math is the hidden secret to understanding the world

"... înțelegerea constă în abilitatea de a-ți schimba perspectiva"

[https://www.ted.com/talks/roger\\_antonsen\\_math\\_is\\_the\\_hidden\\_secret\\_to\\_understanding\\_the\\_world](https://www.ted.com/talks/roger_antonsen_math_is_the_hidden_secret_to_understanding_the_world)

## Un program simplu în Haskell

```
data Point = Point Int Int
```

```
makePoint :: Int -> Int -> Point
```

```
makePoint x y = Point x y
```

```
getX :: Point -> Int
```

```
getX (Point x y) = x
```

```
getY :: Point -> Int
```

```
getY (Point x y) = y
```

```
origin :: Point
```

```
origin = makePoint 0 0
```

# Un program simplu în Haskell

Hai să schimbăm perspectiva!

**data** Point = Point **Int Int**

**makePoint** :: **Int** -> **Int** -> Point      $\frac{x : \text{Int} \quad y : \text{Int}}{\text{makePoint } x \ y : \text{Point}} \text{ (Point}_I\text{)}$   
**makePoint** x y = Point x y

**getX** :: Point -> **Int**      $\frac{p : \text{Point}}{\text{getX } p : \text{Int}} \text{ (Point}_{E_1}\text{)}$   
**getX** (Point x y) = x

**getY** :: Point -> **Int**      $\frac{p : \text{Point}}{\text{getY } p : \text{Int}} \text{ (Point}_{E_2}\text{)}$   
**getY** (Point x y) = y

$$\frac{x : \text{Int} \quad y : \text{Int}}{\text{makePoint } x \ y : \text{Point}} (\text{Point}_I)$$

$$\frac{M : \sigma \quad N : \tau}{\langle M, N \rangle : \sigma \times \tau} (\times_I)$$

$$\frac{p : \text{Point}}{\text{getX } p : \text{Int}} (\text{Point}_{E_1})$$

$$\frac{M : \sigma \times \tau}{\text{fst } M : \sigma} (\times_{E_1})$$

$$\frac{p : \text{Point}}{\text{getY } p : \text{Int}} (\text{Point}_{E_2})$$

$$\frac{M : \sigma \times \tau}{\text{snd } M : \tau} (\times_{E_2})$$

$$\frac{x : \text{Int} \quad y : \text{Int}}{\text{makePoint } x \ y : \text{Point}} \text{ (Point}_I\text{)}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau} (\times_I)$$

$$\frac{M : \text{Point}}{\text{getX } M : \text{Int}} \text{ (Point}_{E_1}\text{)}$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{fst } M : \sigma} (\times_{E_1})$$

$$\frac{M : \text{Point}}{\text{getY } M : \text{Int}} \text{ (Point}_{E_2}\text{)}$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{snd } M : \tau} (\times_{E_2})$$



## Alt exemplu simplu

$f = (\lambda x \rightarrow x * 3) :: \text{Int} \rightarrow \text{Int}$

$$\frac{\{x : \text{Int}\} \vdash x * 3 : \text{Int}}{\lambda x. x * 3 : \text{Int} \rightarrow \text{Int}} \text{ (fun}_I\text{)}$$

$> f \ 5$   
 $15$

$$\frac{f : \text{Int} \rightarrow \text{Int} \quad 5 : \text{Int}}{f \ 5 : \text{Int}} \text{ (fun}_E\text{)}$$

# Generalizare

$$\frac{\{x : \text{Int}\} \vdash x * 3 : \text{Int}}{\lambda x. x * 3 : \text{Int} \rightarrow \text{Int}} \text{ (fun}_I\text{)}$$

$$\frac{\{x : \sigma\} \vdash M : \tau}{\lambda x. M : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{f : \text{Int} \rightarrow \text{Int} \quad 5 : \text{Int}}{f \ 5 : \text{Int}} \text{ (fun}_E\text{)}$$

$$\frac{M : \sigma \rightarrow \tau \quad N : \sigma}{MN : \tau} (\rightarrow_E)$$

# Generalizare

$$\frac{\{x : \text{Int}\} \vdash x * 3 : \text{Int}}{\lambda x. x * 3 : \text{Int} \rightarrow \text{Int}} \text{ (fun}_I\text{)}$$

$$\frac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x. M : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{f : \text{Int} \rightarrow \text{Int} \quad 5 : \text{Int}}{f \ 5 : \text{Int}} \text{ (fun}_E\text{)}$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

## Logica. Ce este adevărat și ce este fals?

Hai să schimbăm perspectiva iar!

## Logica. Ce este adevărat și ce este fals?

Hai să schimbăm perspectiva iar!

Dacă afară este întuneric atunci,  
dacă porcii zboară atunci este întuneric afară.

$\sigma$  = afară este întuneric

$\tau$  = porcii zboară

$\sigma \supset (\tau \supset \sigma)$

# Logica. Ce este adevărat și ce este fals?

Hai să schimbăm perspectiva iar!

Dacă afară este întuneric atunci,  
dacă porcii zboară atunci este întuneric afară.

$\sigma$  = afară este întuneric  
 $\tau$  = porcii zboară

$$\sigma \supset (\tau \supset \sigma)$$

Este adevărată această afirmație? Da!

$\sigma$	$\tau$	$\tau \supset \sigma$	$\sigma \supset (\tau \supset \sigma)$
false	false	true	true
false	true	false	true
true	false	true	true
true	true	true	true

## Semantica unei logici

Dăm valori variabilelor în mulțimea  $\{0, 1\}$ ,  
definim o evaluare  $e : V \rightarrow \{0, 1\}$ .

Putem să o extindem o evaluare la formule:

$$\wedge : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

$\sigma$	$\tau$	$\sigma \wedge \tau$
0	0	0
0	1	0
1	0	0
1	1	1

$$\supset : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

$\sigma$	$\tau$	$\sigma \supset \tau$
0	0	1
0	1	1
1	0	0
1	1	1

Dacă pentru toate evaluările posibile, o formulă are valoarea 1,  
atunci spunem că este o **tautologie**.

Dăm metode pentru a manipula simbolurile din logică (i.e.,  $\supset$ ,  $\wedge$ ) pentru a stabili când o formulă este **demonstrabilă/teoremă** .

**Corectitudine** = sintaxa implică semantica  
**Completitudine** = sintaxa și semantica coincid



# Un sistem de deducție naturală

Reguli pentru a manevra fiecare conector logic  
(introducerea si eliminarea conectorilor).

$$\frac{\Gamma \vdash \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma \wedge \tau} (\wedge_I)$$

$$\frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \sigma} (\wedge_{E_1})$$

$$\frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \tau} (\wedge_{E_2})$$

$$\frac{\Gamma \cup \{\sigma\} \vdash \tau}{\Gamma \vdash \sigma \supset \tau} (\supset_I)$$

$$\frac{\Gamma \vdash \sigma \supset \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau} (\supset_E)$$

Arată cunoscut?

# Corespondența Curry-Howard

## $\lambda$ -calcul cu tipuri

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau} (\times_I)$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{fst } M : \sigma} (\times_{E_1})$$

$$\frac{\Gamma \vdash p : \sigma \times \tau}{\Gamma \vdash \text{snd } p : \tau} (\times_{E_2})$$

$$\frac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x. M : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau} (\rightarrow_E)$$

## Deducție naturală

$$\frac{\Gamma \vdash \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma \wedge \tau} (\wedge_I)$$

$$\frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \sigma} (\wedge_{E_1})$$

$$\frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \tau} (\wedge_{E_2})$$

$$\frac{\Gamma \cup \{\sigma\} \vdash \tau}{\Gamma \vdash \sigma \supset \tau} (\supset_I)$$

$$\frac{\Gamma \vdash \sigma \supset \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau} (\supset_E)$$

*Propositions are types! ♥*

## Să analizăm mai atent

$\lambda$ -calcul cu tipuri      Deducție naturală

$\Gamma \vdash M : \sigma$

$\Gamma \vdash \sigma$

Faptul că există un termen de tip  $\sigma$  (*inhabitation of type  $\sigma$* )  
înseamnă că  $\sigma$  este teoremă/are o demonstrație în logică! ♥

## Să analizăm mai atent

$\lambda$ -calcul cu tipuri

$$\frac{\{x:\sigma\} \vdash x:\sigma}{\vdash \lambda x. x:\sigma \rightarrow \sigma} (\rightarrow_I)$$

Deducție naturală

$$\frac{\{\sigma\} \vdash \sigma}{\vdash \sigma \supset \sigma} (\supset_I)$$

## $\lambda$ -calcul cu tipuri

$$\frac{\{x:\sigma\} \vdash x:\sigma}{\vdash \lambda x. x:\sigma \rightarrow \sigma} (\rightarrow_I)$$

$$\frac{\frac{\overline{\{x:\sigma, y:\tau\} \vdash x:\sigma}}{\{x:\sigma\} \vdash \lambda y. x:\tau \rightarrow \sigma} (\rightarrow_I)}{\vdash \lambda x. (\lambda y. x):\sigma \rightarrow (\tau \rightarrow \sigma)} (\rightarrow_I)$$

## Deducție naturală

$$\frac{\{\sigma\} \vdash \sigma}{\vdash \sigma \supset \sigma} (\supset_I)$$

$$\frac{\frac{\overline{\{\sigma, \tau\} \vdash \sigma}}{\{\sigma\} \vdash \tau \rightarrow \sigma} (\supset_I)}{\vdash \sigma \rightarrow (\tau \rightarrow \sigma)} (\supset_I)$$

# Să analizăm mai atent

## $\lambda$ -calcul cu tipuri

$$\frac{\{x:\sigma\} \vdash x:\sigma}{\vdash \lambda x. x:\sigma \rightarrow \sigma} (\rightarrow_I)$$

$$\frac{\frac{\overline{\{x:\sigma, y:\tau\} \vdash x:\sigma}}{\{x:\sigma\} \vdash \lambda y. x:\tau \rightarrow \sigma} (\rightarrow_I)}{\vdash \lambda x. (\lambda y. x):\sigma \rightarrow (\tau \rightarrow \sigma)} (\rightarrow_I)$$

## Deducție naturală

$$\frac{\{\sigma\} \vdash \sigma}{\vdash \sigma \supset \sigma} (\supset_I)$$

$$\frac{\frac{\overline{\{\sigma, \tau\} \vdash \sigma}}{\{\sigma\} \vdash \tau \rightarrow \sigma} (\supset_I)}{\vdash \sigma \rightarrow (\tau \rightarrow \sigma)} (\supset_I)$$

Proofs are Terms! ♥

Demonstrațiile sunt termeni!

# Correspondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
<i>inhabitation</i> a tipului $\sigma$	demonstrație a lui $\sigma$

# Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
<i>inhabitation</i> a tipului $\sigma$	demonstrație a lui $\sigma$
tip produs	conjunție
tip funcție	implicație



# Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
<i>inhabitation</i> a tipului $\sigma$	demonstrație a lui $\sigma$
tip produs	conjunție
tip funcție	implicație
tip sumă	disjunție
tipul void	false
tipul unit	true

# Logica intuiționistă

- Logică **constructivistă**
- Bazată pe noțiunea de **demonstrație**
- Utilă deoarece demonstrațiile **sunt executabile** și **produc exemple**  
Permite "extragererea" de programe demonstrate a fi corecte.
- Baza pentru *proof assistants* (e.g., Coq, Agda, Idris)
- **Următoarele formule echivalente nu sunt demonstrabile în logica intuiționistă!**
  - dubla negație:  $\neg\neg\varphi \supset \varphi$
  - excluded middle:  $\varphi \vee \neg\varphi$
  - legea lui Pierce:  $((\varphi \supset \tau) \supset \varphi) \supset \varphi$
- **Nu există semantică cu tabele de adevăr pentru logica intuiționistă!** Semantici alternative (e.g., semantica de tip Kripke)

Inițial, corespondența Curry-Howard a fost între

Calculul  
Church  $\lambda \rightarrow$

Sistemul de deducție naturală  
al lui Gentzen pentru  
logica intuiționistă

- Este pur si simplu fascinant
- Nu gândiți logica și informatica ca domenii diferite.
- Gândind din perspective diferite ne poate ajuta să știm ce este posibil/imposibil.
- Teoria tipurilor nu ar trebui să fie o adunătură *ad hoc* de reguli!

Quiz time!



<https://tinyurl.com/C06-Quiz1>

**Pe săptămâna viitoare!**

# Fundamentele limbajelor de programare

C07

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## **Lambda calcul cu tipuri simple (cont.)**

---



# Ce problemă vrem să rezolvăm în cursul de astăzi?

## Type Inference

Pentru un lambda termen  $M$  fără tipuri, vrem să adnotăm termenul  $M$  cu tipuri obținând  $\overline{M}$  și să rezolvăm problema

$$? \vdash \overline{M} : ?$$

(să găsim un context și un tip, pentru a avea o judecată legală).

## Exemple:

- Pentru termenul  $(\lambda z. \lambda u. z) (y x)$ , am putea obține

$$\{x : \alpha, y : \alpha \rightarrow \beta\} \vdash (\lambda z : \beta. \lambda u : \gamma. z) (y x) : \gamma \rightarrow \beta$$

- Pentru termenul  $x x$  nu putem să rezolvăm problema.

# Tipuri simple

Sistemul  $\lambda \rightarrow$

$\Gamma \vdash M : \sigma$

$\overline{\Gamma \vdash x : \sigma} \text{ (var) dacă } x : \sigma \in \Gamma$

$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$

$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$

# Tipuri simple

## Sistemul $\lambda \rightarrow$

$$\Gamma \vdash M : \sigma$$

$$\frac{}{\Gamma \vdash x : \sigma} \text{ (var) } \text{dacă } x : \sigma \in \Gamma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

## Sistemul $\lambda \rightarrow$ cu constrângeri

# Tipuri simple

## Sistemul $\lambda \rightarrow$

$$\Gamma \vdash M : \sigma$$

$$\frac{}{\Gamma \vdash x : \sigma} \text{ (var) } \text{dacă } x : \sigma \in \Gamma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

## Sistemul $\lambda \rightarrow$ cu constrângeri

$$\Gamma \vdash M : \sigma \triangleright C$$

# Tipuri simple

## Sistemul $\lambda \rightarrow$

$$\Gamma \vdash M : \sigma$$

$$\frac{}{\Gamma \vdash x : \sigma} \text{ (var) } \text{dacă } x : \sigma \in \Gamma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

## Sistemul $\lambda \rightarrow$ cu constrângeri

$$\Gamma \vdash M : \sigma \triangleright C$$

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \sigma \triangleright \{\sigma = \tau\}} \text{ (var*)}$$

# Tipuri simple

## Sistemul $\lambda \rightarrow$

$$\Gamma \vdash M : \sigma$$

$$\frac{}{\Gamma \vdash x : \sigma} (var) \text{ dacă } x : \sigma \in \Gamma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

## Sistemul $\lambda \rightarrow$ cu constrângeri

$$\Gamma \vdash M : \sigma \triangleright C$$

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \sigma \triangleright \{\sigma \dot{=} \tau\}} (var^*)$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau' \triangleright C' \quad C = C' \cup \{\tau \dot{=} \sigma \rightarrow \tau'\}}{\Gamma \vdash (\lambda x : \sigma. M) : \tau \triangleright C} (\rightarrow_l^*)$$

# Tipuri simple

## Sistemul $\lambda \rightarrow$

$$\Gamma \vdash M : \sigma$$

$$\frac{}{\Gamma \vdash x : \sigma} (var) \text{ dacă } x : \sigma \in \Gamma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

## Sistemul $\lambda \rightarrow$ cu constrângeri

$$\Gamma \vdash M : \sigma \triangleright C$$

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \sigma \triangleright \{\sigma \dot{=} \tau\}} (var^*)$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau' \triangleright C' \quad C = C' \cup \{\tau \dot{=} \sigma \rightarrow \tau'\}}{\Gamma \vdash (\lambda x : \sigma. M) : \tau \triangleright C} (\rightarrow_I^*)$$

$$\frac{\Gamma \vdash M : \tau_1 \triangleright C_1 \quad \Gamma \vdash N : \tau_2 \triangleright C_2 \quad C = C_1 \cup C_2 \cup \{\tau_1 \dot{=} \tau_2 \rightarrow \tau\}}{\Gamma \vdash MN : \tau \triangleright C} (\rightarrow_E^*)$$

# Tipuri simple

## Sistemul $\lambda \rightarrow$

$$\Gamma \vdash M : \sigma$$

$$\frac{}{\Gamma \vdash x : \sigma} (var) \text{ dacă } x : \sigma \in \Gamma$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma. M) : \sigma \rightarrow \tau} (\rightarrow_I)$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\rightarrow_E)$$

$\sigma, \tau$  variabile de tip

## Sistemul $\lambda \rightarrow$ cu constrângeri

$$\Gamma \vdash M : \sigma \triangleright C$$

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \sigma \triangleright \{\sigma \doteq \tau\}} (var^*)$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau' \triangleright C' \quad C = C' \cup \{\tau \doteq \sigma \rightarrow \tau'\}}{\Gamma \vdash (\lambda x : \sigma. M) : \tau \triangleright C} (\rightarrow_I^*)$$

$$\frac{\Gamma \vdash M : \tau_1 \triangleright C_1 \quad \Gamma \vdash N : \tau_2 \triangleright C_2 \quad C = C_1 \cup C_2 \cup \{\tau_1 \doteq \tau_2 \rightarrow \tau\}}{\Gamma \vdash MN : \tau \triangleright C} (\rightarrow_E^*)$$

$\sigma, \tau, \tau', \tau_1, \tau_2$  variabile de tip



## Sistemul $\lambda \rightarrow$ cu constrângeri

O judecată de forma  $\Gamma \vdash M : \sigma \triangleright C$  este **legală** dacă constrângerile din  $C$  au o "soluție".

De exemplu, judecata de mai jos este legală

$$\{x : \alpha, y : \alpha \rightarrow \beta\} \vdash (\lambda z : \beta. \lambda u : \gamma. z) (y \ x) : \gamma \rightarrow \beta \triangleright C, \text{ unde} \\ C = \{\delta \doteq \beta, \tau'_1 \doteq (\gamma \rightarrow \delta), \tau_1 \doteq (\beta \rightarrow \tau'_1), \sigma_1 \doteq \alpha \rightarrow \beta, \sigma_2 \doteq \alpha, \\ \sigma_1 \doteq \sigma_2 \rightarrow \tau_2, \tau_1 \doteq (\tau_2 \rightarrow (\gamma \rightarrow \beta))\}$$

$C$  are "soluție" și spune, de exemplu, că ar trebui ca  $\alpha = \sigma_2$  și  $\beta = \delta$ .

În slide-urile următoare dăm mai multe detalii despre ce înseamnă acest lucru.

# Type Inference

Fie  $M$  un lambda termen fără tipuri.

Construim un context  $\Gamma_M$  pentru  $M$ :

$$\Gamma_M = \{x : X \mid x \in FV(M)\}$$

(toate variabilele de tip  $X$  introduse mai sus sunt noi și distincte)

Adnotăm  $M$  cu tipuri pentru variabilele legate obținând  $\overline{M}$  prin inducție după structura lui  $M$  astfel:

- dacă  $M = x$ , atunci  $\overline{M} = M$
- dacă  $M = M_1 N_1$ , atunci  $\overline{M} = \overline{M_1} \overline{N_1}$
- dacă  $M = \lambda x. N$ , atunci  $\overline{M} = \lambda x : X. \overline{N}$ , unde  $X$  este o variabilă de tip nouă

Fie  $M$  un lambda termen fără tipuri.

Dacă există o constrângere de tipuri  $C_M$  și o variabilă de tip nouă  $V$  astfel încât

$$\Gamma_M \vdash \overline{M} : V \triangleright C_M$$

este o judecată legală, atunci  $M$  este *typable*.  
(soluția o găsim prin  $C_M$ )

# Type Inference - Exemplul 1

Fie  $M_1 = (\lambda z. \lambda u. z) (y x)$ .

Obținem  $\Gamma_{M_1} = \{x: X, y: Y\}$  și  $\overline{M_1} = (\lambda z: Z. \lambda u: U. z) (y x)$ .

$$\begin{array}{c}
 \Gamma_{M_1} \cup \{z: Z, u: U\} \vdash z: \delta \triangleright D \\
 C'_1 = D \cup \{\tau'_1 \doteq (U \rightarrow \delta)\} \\
 \hline
 \Gamma_{M_1} \cup \{z: Z\} \vdash \lambda u: U. z: \tau'_1 \triangleright C'_1 \quad (\rightarrow_I^*) \\
 C_1 = C'_1 \cup \{\tau_1 \doteq (Z \rightarrow \tau'_1)\} \\
 \hline
 \Gamma_{M_1} \vdash \lambda z: Z. \lambda u: U. z: \tau_1 \triangleright C_1 \quad (\rightarrow_I^*) \\
 \\
 \Gamma_{M_1} \vdash y: \sigma_1 \triangleright C'_2 \quad \Gamma_{M_1} \vdash x: \sigma_2 \triangleright C''_2 \\
 C_2 = C'_2 \cup C''_2 \cup \{\sigma_1 \doteq \sigma_2 \rightarrow \tau_2\} \\
 \hline
 \Gamma_{M_1} \vdash y x: \tau_2 \triangleright C_2 \quad (\rightarrow_E^*) \\
 \\
 C_{M_1} = C_1 \cup C_2 \cup \{\tau_1 \doteq (\tau_2 \rightarrow V)\} \\
 \hline
 \Gamma_{M_1} \vdash (\lambda z: Z. \lambda u: U. z) (y x): V \triangleright C_{M_1}
 \end{array}$$

$$D = \{\delta \doteq Z\}$$

$$C'_1 = \{\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta)\}$$

$$C_1 = \{\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta), \tau_1 \doteq (Z \rightarrow \tau'_1)\}$$

$$C'_2 = \{\sigma_1 \doteq Y\}$$

$$C''_2 = \{\sigma_2 \doteq X\}$$

$$C_2 = \{\sigma_1 \doteq Y, \sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2\}$$

$$C_{M_1} = \{\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta), \tau_1 \doteq (Z \rightarrow \tau'_1), \sigma_1 \doteq Y, \sigma_2 \doteq X,$$

$$\sigma_1 \doteq \sigma_2 \rightarrow \tau_2, \tau_1 \doteq (\tau_2 \rightarrow V)\}$$

Constrângerile  $C_{M_1}$  au "soluție". Ce înseamnă asta?

## Type Inference - Exemplul 2

Fie  $M_2 = x\ x$ .

Obținem  $\Gamma_{M_2} = \{x:X\}$  și  $\overline{M_2} = M_2$ .

$$\frac{\begin{array}{l} \{x:X\} \vdash x:\tau_1 \triangleright C_1 \quad \{x:X\} \vdash x:\tau_2 \triangleright C_2 \\ C_M = C_1 \cup C_2 \cup \{\tau_1 \dot{=} \tau_2 \rightarrow V\} \end{array}}{\{x:X\} \vdash (x\ x): V \triangleright C_{M_2}} \quad (\rightarrow_E^*)$$

$$C_1 = \{\tau_1 \dot{=} X\}$$

$$C_2 = \{\tau_2 \dot{=} X\}$$

$$C_{M_2} = \{\tau_1 \dot{=} X, \tau_2 \dot{=} X, \tau_1 \dot{=} \tau_2 \rightarrow V\}$$

Constrângerile  $C_{M_2}$  nu au "soluție". Ce înseamnă asta?

## Type Inference - Exemplul 2

Fie  $M_2 = x\ x$ .

Obținem  $\Gamma_{M_2} = \{x:X\}$  și  $\overline{M_2} = M_2$ .

$$\frac{\begin{array}{l} \{x:X\} \vdash x:\tau_1 \triangleright C_1 \quad \{x:X\} \vdash x:\tau_2 \triangleright C_2 \\ C_M = C_1 \cup C_2 \cup \{\tau_1 \doteq \tau_2 \rightarrow V\} \end{array}}{\{x:X\} \vdash (x\ x):V \triangleright C_{M_2}} \quad (\rightarrow_E^*)$$

$$C_1 = \{\tau_1 \doteq X\}$$

$$C_2 = \{\tau_2 \doteq X\}$$

$$C_{M_2} = \{\tau_1 \doteq X, \tau_2 \doteq X, \tau_1 \doteq \tau_2 \rightarrow V\}$$

Constrângerile  $C_{M_2}$  nu au "soluție". Ce înseamnă asta?

Constrângerile au "soluție" dacă se pot **unifica**.

# Unificare

---

## Alfabet:

- $\mathcal{F}$  o mulțime de simboluri de funcții de aritate cunoscută
- $\mathcal{V}$  o mulțime numărabilă de variabile
- $\mathcal{F}$  și  $\mathcal{V}$  sunt disjuncte

## Termeni peste $\mathcal{F}$ și $\mathcal{V}$ :

$$t ::= x \mid f(t_1, \dots, t_n)$$

- $n \geq 0$
- $x$  este o variabilă
- $f$  este un simbol de funcție de aritate  $n$

Pentru ușurință, considerăm funcțiile în forma prefixată.



## Notatii:

- **constante**: simboluri de funcții de aritate 0
- $x, y, z, \dots$  pentru variabile
- $a, b, c, \dots$  pentru constante
- $f, g, h, \dots$  pentru simboluri de funcții arbitrare
- $s, t, u, \dots$  pentru termeni
- $\text{var}(t)$  mulțimea variabilelor care apar în  $t$
- ecuații  $s \doteq t$  pentru o pereche de termeni
- $\text{Trm}_{\mathcal{F}, \mathcal{V}}$  mulțimea termenilor peste  $\mathcal{F}$  și  $\mathcal{V}$

## Exemple:

- $f(x, g(x, a), y)$  este un termen, unde  $f$  are aritate 3,  $g$  are aritate 2,  $a$  este o constanta
- $\text{var}(f(x, g(x, a), y)) = \{x, y\}$

# Legătura cu teoria tipurilor

Mulțimea **tipurilor simple**  $\mathbb{T} = \mathbb{V} \mid \mathbb{T} \rightarrow \mathbb{T}$

În acest caz, avem alfabetul:

- $\mathcal{F} = \{\rightarrow\}$ , iar aritatea lui  $\rightarrow$  este 2
- $\mathcal{V} = \mathbb{V}$

Dacă avem și alte tipuri, extindem  $\mathcal{F}$  cu noi simboluri. De exemplu,

- `Unit`, `Void` cu aritate 0 (deci constante)
- `Bool`, `Nat` cu aritate 0 (deci constante)
- `Maybe`, `List` cu aritate 1
- $\times$  cu aritate 2
- ...

O substituție  $\Theta$  este o funcție (parțială) de la variabile la termeni,

$$\Theta : \mathcal{V} \rightarrow \text{Trm}_{\mathcal{F}, \mathcal{V}}$$

Exemplu:

În notația uzuală,  $\Theta = \{a/x, g(w)/y, b/z\}$ .

Substituția  $\Theta$  este identitate pe restul variabilelor.

Notatie alternativă  $\Theta = \{x \mapsto a, y \mapsto g(w), z \mapsto b\}$ .

# Substituții

Substituțiile sunt o modalitate de a înlocui variabile cu alți termeni.

Substituțiile se aplică simultan pe toate variabilele.

Aplicarea unei substituții  $\Theta$  unui termen  $t$ :

$$\Theta(t) = \begin{cases} \Theta(x), & \text{dacă } t = x \\ f(\Theta(t_1), \dots, \Theta(t_n)), & \text{dacă } t = f(t_1, \dots, t_n) \end{cases}$$

Exemplu:

- $\Theta = \{x \mapsto f(x, y), y \mapsto g(a)\}$
- $t = f(x, g(f(x, f(y, z))))$
- $\Theta(t) = f(f(x, y), g(f(f(x, y), f(g(a), z))))$

# Substituții

Două substituții  $\Theta_1$  și  $\Theta_2$  se pot compune

$$\Theta_1; \Theta_2$$

(aplicăm întâi  $\Theta_1$ , apoi  $\Theta_2$ ).

Exemplu:

- $t = h(u, v, x, y, z)$
- $\Theta_1 = \{x \mapsto f(y), y \mapsto f(a), z \mapsto u\}$
- $\Theta_2 = \{y \mapsto g(a), u \mapsto z, v \mapsto f(f(a))\}$
- $(\Theta_1; \Theta_2)(t) = \Theta_2(\Theta_1(t)) = \Theta_2(h(u, v, f(y), f(a), u)) =$   
 $= h(z, f(f(a)), f(g(a)), f(a), z)$
- $(\Theta_2; \Theta_1)(t) = \Theta_1(\Theta_2(t)) = \Theta_1(h(z, f(f(a)), x, g(a), z)) =$   
 $= h(u, f(f(a)), f(y), g(a), u)$

Doi termeni  $t_1$  și  $t_2$  **se unifică** dacă există o substituție  $\Theta$  astfel încât

$$\Theta(t_1) = \Theta(t_2).$$

În acest caz,  $\Theta$  se numește un **unificator** al termenilor  $t_1$  și  $t_2$ .

Un unificator  $\Theta$  pentru  $t_1$  și  $t_2$  este **cel mai general unificator** (**cmgu, mgu**) dacă pentru orice alt unificator  $\Theta'$  pentru  $t_1$  și  $t_2$ , există o substituție  $\Delta$  astfel încât

$$\Theta' = \Theta; \Delta.$$

## Exemplu:

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$
- $\Theta = \{x \mapsto y\}$ 
  - $\Theta(t) = y + (y * y)$
  - $\Theta(t') = y + (y * y)$
  - $\Theta$  este **cmgu**
- $\Theta' = \{x \mapsto 0, y \mapsto 0\}$ 
  - $\Theta'(t) = 0 + (0 * 0)$
  - $\Theta'(t') = 0 + (0 * 0)$
  - $\Theta' = \Theta; \{y \mapsto 0\}$
  - $\Theta'$  este **unificator**, dar nu este **cmgu**



Quiz time!



<https://tinyurl.com/C07-Quiz1>

**Vacanță plăcută!**

# Fundamentele limbajelor de programare

C08

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## **Lambda calcul cu tipuri simple și unificare (recap)**

---

## Ce problemă am rezolvat în cursul trecut?

### *Type Inference*

Pentru un lambda termen  $M$  fără tipuri, am adnotat termenul  $M$  cu tipuri obținând  $\overline{M}$  și am rezolvat (parțial) problema

$$? \vdash \overline{M} : ?$$

(am găsit un context și un tip, pentru a avea o judecată legală).

# Type Inference

Fie  $M$  un lambda termen fără tipuri.

Construim un context  $\Gamma_M$  pentru  $M$ :

$$\Gamma_M = \{x : X \mid x \in FV(M)\}$$

(toate variabilele de tip  $X$  introduse mai sus sunt noi și distincte)

Adnotăm  $M$  cu tipuri pentru variabilele legate obținând  $\overline{M}$  prin inducție după structura lui  $M$  astfel:

- dacă  $M = x$ , atunci  $\overline{M} = M$
- dacă  $M = M_1 N_1$ , atunci  $\overline{M} = \overline{M_1} \overline{N_1}$
- dacă  $M = \lambda x. N$ , atunci  $\overline{M} = \lambda x : X. \overline{N}$ , unde  $X$  este o variabilă de tip nouă

## Sistemul $\lambda \rightarrow$ cu constrângeri

$$\Gamma \vdash M : \sigma \triangleright C$$

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \sigma \triangleright \{\sigma \dot{=} \tau\}} \text{ (var}^*)$$

$$\frac{\begin{array}{l} \Gamma, x : \sigma \vdash M : \tau' \triangleright C' \\ C = C' \cup \{\tau \dot{=} \sigma \rightarrow \tau'\} \end{array}}{\Gamma \vdash (\lambda x : \sigma. M) : \tau \triangleright C} (\rightarrow_I^*)$$

$$\frac{\begin{array}{l} \Gamma \vdash M : \tau_1 \triangleright C_1 \quad \Gamma \vdash N : \tau_2 \triangleright C_2 \\ C = C_1 \cup C_2 \cup \{\tau_1 \dot{=} \tau_2 \rightarrow \tau\} \end{array}}{\Gamma \vdash MN : \tau \triangleright C} (\rightarrow_E^*)$$

$\sigma, \tau, \tau', \tau_1, \tau_2$  variabile de tip

## Sistemul $\lambda \rightarrow$ cu constrângeri

O judecată de forma  $\Gamma \vdash M : \sigma \triangleright C$  este **legală** dacă constrângerile din  $C$  au o "soluție".

Fie  $M$  un lambda termen fără tipuri. Dacă există o constrângere de tipuri  $C_M$  și o variabilă de tip nouă  $V$  astfel încât

$$\Gamma_M \vdash \overline{M} : V \triangleright C_M$$

este o judecată legală, atunci  $M$  este *typable*.  
(soluția o găsim prin  $C_M$ )



# Type Inference - Exemplul 1

Fie  $M_1 = (\lambda z. \lambda u. z) (y x)$ .

Obținem  $\Gamma_{M_1} = \{x:X, y:Y\}$  și  $\overline{M_1} = (\lambda z:Z. \lambda u:U. z) (y x)$ .

$\Gamma_{M_1} \cup \{z:Z, u:U\} \vdash z:\delta \triangleright D$

$C'_1 = D \cup \{\tau'_1 \doteq (U \rightarrow \delta)\}$

$\frac{\Gamma_{M_1} \cup \{z:Z\} \vdash \lambda u:U. z:\tau'_1 \triangleright C'_1}{\Gamma_{M_1} \cup \{z:Z\} \vdash \lambda u:U. z:\tau'_1 \triangleright C'_1} (\rightarrow^*_I)$

$C_1 = C'_1 \cup \{\tau_1 \doteq (Z \rightarrow \tau'_1)\}$

$\frac{\Gamma_{M_1} \vdash \lambda z:Z. \lambda u:U. z:\tau_1 \triangleright C_1}{\Gamma_{M_1} \vdash \lambda z:Z. \lambda u:U. z:\tau_1 \triangleright C_1} (\rightarrow^*_I)$

$\Gamma_{M_1} \vdash y:\sigma_1 \triangleright C'_2 \quad \Gamma_{M_1} \vdash x:\sigma_2 \triangleright C''_2$

$C_2 = C'_2 \cup C''_2 \cup \{\sigma_1 \doteq \sigma_2 \rightarrow \tau_2\}$

$\frac{\Gamma_{M_1} \vdash y x:\tau_2 \triangleright C_2}{\Gamma_{M_1} \vdash y x:\tau_2 \triangleright C_2} (\rightarrow^*_E)$

$C_{M_1} = C_1 \cup C_2 \cup \{\tau_1 \doteq (\tau_2 \rightarrow V)\}$

$\Gamma_{M_1} \vdash (\lambda z:Z. \lambda u:U. z) (y x): V \triangleright C_{M_1}$

$D = \{\delta \doteq Z\}$

$C'_1 = \{\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta)\}$

$C_1 = \{\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta), \tau_1 \doteq (Z \rightarrow \tau'_1)\}$

$C'_2 = \{\sigma_1 \doteq Y\}$

$C''_2 = \{\sigma_2 \doteq X\}$

$C_2 = \{\sigma_1 \doteq Y, \sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2\}$

$C_{M_1} = \{\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta), \tau_1 \doteq (Z \rightarrow \tau'_1), \sigma_1 \doteq Y, \sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2, \tau_1 \doteq (\tau_2 \rightarrow V)\}$

Constrângerile  $C_{M_1}$  au "soluție". Ce înseamnă asta?

## Type Inference - Exemplul 2

Fie  $M_2 = x \ x$ .

Obținem  $\Gamma_{M_2} = \{x:X\}$  și  $\overline{M_2} = M_2$ .

$$\frac{\begin{array}{l} \{x:X\} \vdash x:\tau_1 \triangleright C_1 \quad \{x:X\} \vdash x:\tau_2 \triangleright C_2 \\ C_M = C_1 \cup C_2 \cup \{\tau_1 \doteq \tau_2 \rightarrow V\} \end{array}}{\{x:X\} \vdash (x \ x): V \triangleright C_{M_2}} \quad (\rightarrow_E^*)$$

$$C_1 = \{\tau_1 \doteq X\}$$

$$C_2 = \{\tau_2 \doteq X\}$$

$$C_{M_2} = \{\tau_1 \doteq X, \tau_2 \doteq X, \tau_1 \doteq \tau_2 \rightarrow V\}$$

Constrângerile  $C_{M_2}$  nu au "soluție". Ce înseamnă asta?

Constrângerile au "soluție" dacă se pot **unifica**.

## Alfabet:

- $\mathcal{F}$  o mulțime de simboluri de funcții de aritate cunoscută
- $\mathcal{V}$  o mulțime numărabilă de variabile
- $\mathcal{F}$  și  $\mathcal{V}$  sunt disjuncte

## Termeni peste $\mathcal{F}$ și $\mathcal{V}$ :

$$t ::= x \mid f(t_1, \dots, t_n)$$

- $n \geq 0$
- $x$  este o variabilă
- $f$  este un simbol de funcție de aritate  $n$

## Notatii:

- **constante**: simboluri de funcții de aritate 0
- $x, y, z, \dots$  pentru variabile
- $a, b, c, \dots$  pentru constante
- $f, g, h, \dots$  pentru simboluri de funcții arbitrare
- $s, t, u, \dots$  pentru termeni
- $\text{var}(t)$  mulțimea variabilelor care apar în  $t$
- ecuații  $s \doteq t$  pentru o pereche de termeni
- $\text{Trm}_{\mathcal{F}, \mathcal{V}}$  mulțimea termenilor peste  $\mathcal{F}$  și  $\mathcal{V}$

# Legătura cu teoria tipurilor

Mulțimea **tipurilor simple**  $\mathbb{T} = \mathbb{V} \mid \mathbb{T} \rightarrow \mathbb{T}$

În acest caz, avem alfabetul:

- $\mathcal{F} = \{\rightarrow\}$ , iar aritatea lui  $\rightarrow$  este 2
- $\mathcal{V} = \mathbb{V}$

Dacă avem și alte tipuri, extindem  $\mathcal{F}$  cu noi simboluri. De exemplu,

- `Unit`, `Void` cu aritate 0 (deci constante)
- `Bool`, `Nat` cu aritate 0 (deci constante)
- `Maybe`, `List` cu aritate 1
- $\times$  cu aritate 2
- ...

# Substituții

O substituție  $\Theta$  este o funcție de la variabile la termeni,

$$\Theta : \mathcal{V} \rightarrow \text{Trm}_{\mathcal{F}, \mathcal{V}}$$

Notăție pentru substituții care schimbă un număr finit de variabile:

$$[u_1/x_1, u_2/x_2, \dots, u_n/x_n]$$

Aplicarea unei substituții  $\Theta$  unui termen  $t$ :

$$\Theta(t) = \begin{cases} \Theta(x), & \text{dacă } t = x \\ f(\Theta(t_1), \dots, \Theta(t_n)), & \text{dacă } t = f(t_1, \dots, t_n) \end{cases}$$

Notăție pentru  $\Theta(t)$ :  $t[u_1/x_1, u_2/x_2, \dots, u_n/x_n]$

Doi termeni  $t_1$  și  $t_2$  **se unifică** dacă există o substituție  $\Theta$  astfel încât

$$\Theta(t_1) = \Theta(t_2).$$

În acest caz,  $\Theta$  se numește un **unificator** al termenilor  $t_1$  și  $t_2$ .

Un unificator  $\Theta$  pentru  $t_1$  și  $t_2$  este **cel mai general unificator** (**cmgu, mgu**) dacă pentru orice alt unificator  $\Theta'$  pentru  $t_1$  și  $t_2$ , există o substituție  $\Delta$  astfel încât

$$\Theta' = \Theta; \Delta.$$

De exemplu, dacă  $\Theta$  este  $[u_1/x_1, u_2/x_2, \dots, u_n/x_n]$ , atunci  $\Delta$  este de forma  $[v_1/y_1, v_2/y_2, \dots, v_m/y_m]$  cu  $x_i \neq y_j$  pentru orice alegere a lui  $i$  și  $j$ , și

$$\Theta' = [\Delta(u_1)/x_1, \dots, \Delta(u_n)/x_n, v_1/y_1, \dots, v_m/y_m]$$

## Exemplu:

- $t = x + (y * y) = +(x, *(y, y))$
- $t' = x + (y * x) = +(x, *(y, x))$
- $\Theta = \{x \mapsto y\}$ 
  - $\Theta(t) = y + (y * y)$
  - $\Theta(t') = y + (y * y)$
  - $\Theta$  este **cmgu**
- $\Theta' = \{x \mapsto 0, y \mapsto 0\}$ 
  - $\Theta'(t) = 0 + (0 * 0)$
  - $\Theta'(t') = 0 + (0 * 0)$
  - $\Theta' = \Theta; \{y \mapsto 0\}$
  - $\Theta'$  este **unificator**, dar nu este **cmgu**



# Algoritmul de unificare

---

# Algoritmul de unificare

- Pentru o mulțime finită de termeni  $\{t_1, \dots, t_n\}$ ,  $n \geq 2$ , **algoritmul de unificare** stabilește dacă există un cmgu.
- Există algoritmi mai eficienți, dar îl alegem pe acesta pentru simplitatea sa.
- Algoritmul lucrează cu două liste:
  - **Lista soluție:**  $S$
  - **Lista de rezolvat:**  $R$
- **Inițial:**
  - **Lista soluție:**  $S = \emptyset$
  - **Lista de rezolvat:**  $R = \{t_1 \dot{=} t_2, \dots, t_{n-1} \dot{=} t_n\}$   
 $\dot{=}$  este un simbol nou care ne ajută să formăm perechi de termeni ("ecuații")

# Algoritmul de unificare

Algoritmul constă în aplicarea regulilor de mai jos:

- **SCOATE**

- orice ecuație de forma  $t \doteq t$  din  $R$  este eliminată.

- **DESCOMPUNE**

- orice ecuație de forma  $f(t_1, \dots, t_n) \doteq f(t'_1, \dots, t'_n)$  din  $R$  este înlocuită cu ecuațiile  $t_1 \doteq t'_1, \dots, t_n \doteq t'_n$ .

- **REZOLVĂ**

- orice ecuație de forma  $x \doteq t$  sau  $t \doteq x$  din  $R$ , unde variabila  $x$  nu apare în termenul  $t$ , este mutată sub forma  $x \doteq t$  în  $S$ .  
În toate celelalte ecuații (din  $R$  și  $S$ ),  $x$  este înlocuit cu  $t$ .

# Algoritmul de unificare

Algoritmul se termină normal dacă  $R = \emptyset$ .

În acest caz,  $S$  conține cmgu.

Algoritmul este oprit cu concluzia inexistenței unui unificator dacă:

1. În  $R$  există o ecuație de forma

$$f(t_1, \dots, t_n) \doteq g(t'_1, \dots, t'_k) \text{ cu } f \neq g.$$

2. În  $R$  există o ecuație de forma  $x \doteq t$  sau  $t \doteq x$  și variabila  $x$  apare în termenul  $t$ .

## Algoritmul de unificare - schemă

	Lista soluție S	Lista de rezolvat R
Inițial	$\emptyset$	$t_1 \doteq t'_1, \dots, t_n \doteq t'_n$
SCOATE	S	$R', t \doteq t$
	S	$R'$
DESCOMPUNE	S	$R', f(t_1, \dots, t_n) \doteq f(t'_1, \dots, t'_n)$
	S	$R', t_1 \doteq t'_1, \dots, t_n \doteq t'_n$
REZOLVĂ	S	$R', x \doteq t$ sau $t \doteq x$ , x nu apare în t
	$x \doteq t, S[t/x]$	$R'[t/x]$
Final	S	$\emptyset$

$S[t/x]$ : în toate ecuațiile din S, x este înlocuit cu t

## Exemplul 1

Ecuatiile  $\{g(y) \doteq x, f(x, h(x), y) \doteq f(g(z), w, z)\}$  au cmgu?

## Exemplul 1

Ecuatiile  $\{g(y) \doteq x, f(x, h(x), y) \doteq f(g(z), w, z)\}$  au cmgu?

S	R	
$\emptyset$	$g(y) \doteq x, f(x, h(x), y) \doteq f(g(z), w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(g(z), w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z), h(g(y)) \doteq w, y \doteq z$	REZOLVĂ
$w \doteq h(g(y)),$ $x \doteq g(y)$	$g(y) \doteq g(z), y \doteq z$	REZOLVĂ
$y \doteq z, x \doteq g(z),$ $w \doteq h(g(z))$	$g(z) \doteq g(z)$	SCOATE
$y \doteq z, x \doteq g(z),$ $w \doteq h(g(z))$	$\emptyset$	

$\Theta = \{y \mapsto z, x \mapsto g(z), w \mapsto h(g(z))\}$  este cmgu.

## Exemplul 2

Ecuatiile  $\{g(y) \doteq x, f(x, h(y), y) \doteq f(g(z), b, z)\}$  au cmgu?



## Exemplul 2

Ecuatiile  $\{g(y) \doteq x, f(x, h(y), y) \doteq f(g(z), b, z)\}$  au cmgu?

S	R	
$\emptyset$	$g(y) \doteq x, f(x, h(y), y) \doteq f(g(z), b, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(y), y) \doteq f(g(z), b, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq g(z), h(y) \doteq b, y \doteq z$	- EȘEC -

- $h$  și  $b$  sunt simboluri de funcții diferite!
- Nu există unificator pentru acești termeni.

## Exemplul 3

Ecuatiile  $\{g(y) \doteq x, f(x, h(x), y) \doteq f(y, w, z)\}$  au cmgu?

## Exemplul 3

Ecuatiile  $\{g(y) \doteq x, f(x, h(x), y) \doteq f(y, w, z)\}$  au cmgu?

S	R	
$\emptyset$	$g(y) \doteq x, f(x, h(x), y) \doteq f(y, w, z)$	REZOLVĂ
$x \doteq g(y)$	$f(g(y), h(g(y)), y) \doteq f(y, w, z)$	DESCOMPUNE
$x \doteq g(y)$	$g(y) \doteq y, h(g(y)) \doteq w, y \doteq z$	- EȘEC -

- În ecuația  $g(y) \doteq y$ , variabila  $y$  apare în termenul  $g(y)$ .
- Nu există unificator pentru aceste ecuații.

## Exemplul 4

Înapoi la constrângerea obținută când am vorbit de *type inference* pentru termenul  $M_1 = (\lambda z. \lambda u. z) (y x)$ .

Am obținut constrângerile

$$C_{M_1} = \{\delta \dot{=} Z, \tau'_1 \dot{=} (U \rightarrow \delta), \tau_1 \dot{=} (Z \rightarrow \tau'_1), \sigma_1 \dot{=} Y, \sigma_2 \dot{=} X, \\ \sigma_1 \dot{=} \sigma_2 \rightarrow \tau_2, \tau_1 \dot{=} (\tau_2 \rightarrow V)\}$$

- $\rightarrow$  simbol de funcție de aritate 2
- $\delta, \tau_1, \tau'_1, \tau_2, \sigma_1, \sigma_2, X, Y, Z, U, V$  variabile

## Exemplul 4 (cont.)

S	R	
$\emptyset$	$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow \delta), \tau_1 \doteq (Z \rightarrow \tau'_1), \sigma_1 \doteq Y$ $\sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2, \tau_1 \doteq (\tau_2 \rightarrow V)$	REZ.
$\delta \doteq Z$	$\tau'_1 \doteq (U \rightarrow Z), \tau_1 \doteq (Z \rightarrow \tau'_1), \sigma_1 \doteq Y$ $\sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2, \tau_1 \doteq (\tau_2 \rightarrow V)$	REZ.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z)$	$\tau_1 \doteq (Z \rightarrow (U \rightarrow Z)), \sigma_1 \doteq Y$ $\sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2, \tau_1 \doteq (\tau_2 \rightarrow V)$	REZ.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z))$	$\sigma_1 \doteq Y, \sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2,$ $(Z \rightarrow (U \rightarrow Z)) \doteq (\tau_2 \rightarrow V)$	DESC.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z))$	$\sigma_1 \doteq Y, \sigma_2 \doteq X, \sigma_1 \doteq \sigma_2 \rightarrow \tau_2,$ $Z \doteq \tau_2, U \rightarrow Z \doteq V$	REZ.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z)),$ $\sigma_1 \doteq Y$	$\sigma_2 \doteq X, Y \doteq \sigma_2 \rightarrow \tau_2,$ $Z \doteq \tau_2, U \rightarrow Z \doteq V$	REZ.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z)),$ $\sigma_1 \doteq Y, \sigma_2 \doteq X$	$Y \doteq X \rightarrow \tau_2,$ $Z \doteq \tau_2, U \rightarrow Z \doteq V$	REZ.

## Exemplul 4 (cont.)

S	R	
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z)),$ $\sigma_1 \doteq Y, \sigma_2 \doteq X, \tau_2 \doteq Z$	$Y \doteq X \rightarrow Z, U \rightarrow Z \doteq V$	REZ.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z)),$ $\sigma_1 \doteq X \rightarrow Z, \sigma_2 \doteq X, \tau_2 \doteq Z$ $Y \doteq X \rightarrow Z$	$U \rightarrow Z \doteq V$	REZ.
$\delta \doteq Z, \tau'_1 \doteq (U \rightarrow Z),$ $\tau_1 \doteq (Z \rightarrow (U \rightarrow Z)),$ $\sigma_1 \doteq X \rightarrow Z, \sigma_2 \doteq X, \tau_2 \doteq Z$ $Y \doteq X \rightarrow Z, V \doteq U \rightarrow Z$		

Constrângerile se pot unifica!

## Exemplul 5

Înapoi la constrângerea obținută când am vorbit de *type inference* pentru termenul  $M_2 = x\ x$ .

Am obținut constrângerile

$$C_{M_2} = \{\tau_1 \doteq X, \tau_2 \doteq X, \tau_1 \doteq \tau_2 \rightarrow V\}$$

- $\rightarrow$  simbol de funcție de aritate 2
- $\tau_1, \tau_2, V$  variabile

## Exemplul 5 (cont.)

$S$	$R$	
$\emptyset$	$\tau_1 \doteq X, \tau_2 \doteq X, \tau_1 \doteq \tau_2 \rightarrow V$	REZ.
$\tau_1 \doteq X$	$\tau_2 \doteq X, X \doteq \tau_2 \rightarrow V$	REZ.
$\tau_1 \doteq X, \tau_2 \doteq X$	$X \doteq X \rightarrow V$	- EȘEC -

- În ecuația  $X \doteq X \rightarrow V$ , variabila  $X$  apare în termenul  $X \rightarrow V$ .
- Nu există unificator pentru aceste ecuații.



Considerăm

- $x, y, z, u, v$  variabile,
- $a, b, c$  simboluri de constantă,
- $h, g$  simboluri de funcție de aritate 1,
- $f$  simbol de funcție de aritate 2,
- $p$  simbol de funcție de aritate 3.

Aplicați algoritmul de unificare de mai sus pentru termenii:

1.  $p(a, x, h(g(y)))$  și  $p(z, h(z), h(u))$
2.  $f(h(a), g(x))$  și  $f(y, y)$
3.  $p(a, x, g(x))$  și  $p(a, y, y)$
4.  $p(x, y, z)$  și  $p(u, f(v, v), u)$

# Exerciții - rezolvări

1.

S	R	
$\emptyset$	$p(a, x, h(g(y))) = p(z, h(z), h(u))$	DESCOMPUNE
$\emptyset$	$a \doteq z, x \doteq h(z), h(g(y)) \doteq h(u)$	REZOLVĂ
$z \doteq a$	$x \doteq h(a), h(g(y)) \doteq h(u)$	REZOLVĂ
$z \doteq a, x \doteq h(a)$	$h(g(y)) \doteq h(u)$	DESCOMPUNE
$z \doteq a, x \doteq h(a)$	$g(y) \doteq u$	REZOLVĂ
$z \doteq a, x \doteq h(a), u \doteq g(y)$	$\emptyset$	

$\Theta = \{z/a, x/h(a), u/g(y)\}$  este cmgu.

## Exerciții - rezolvări

2.

$S$	$R$	
$\emptyset$	$f(h(a), g(x)) \doteq f(y, y)$	DESCOMPUNE
$\emptyset$	$y \doteq h(a), y \doteq g(x)$	REZOLVĂ
$y \doteq h(a)$	$g(x) \doteq h(a)$	EȘEC

Nu există unificator!

## Exerciții - rezolvări

3.

$S$	$R$	
$\emptyset$	$p(a, x, g(x)) \doteq p(a, y, y)$	DESCOMPUNE
$\emptyset$	$a \doteq a, x \doteq y, y \doteq g(x)$	SCOATE
$\emptyset$	$x \doteq y, y \doteq g(x)$	REZOLVĂ
$x \doteq y$	$y \doteq g(y)$	EȘEC

Nu există unificator!

## Exerciții - rezolvări

4.

$S$	$R$	
$\emptyset$	$p(x, y, z) \doteq p(u, f(v, v), u)$	DESCOMPUNE
$\emptyset$	$x \doteq u, y \doteq f(v, v), z \doteq u$	REZOLVĂ
$x \doteq u$	$y \doteq f(v, v), z \doteq u$	REZOLVĂ
$y \doteq f(v, v), x \doteq u$	$z \doteq u$	REZOLVĂ
$z \doteq u, y \doteq f(v, v), x \doteq u$		

$\Theta = \{z/u, y/f(v, v), x/u\}$  este cmgu.

**Pe data viitoare!**

# Fundamentele limbajelor de programare

C09

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

# Programare logică & Prolog

---



# 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 - în mod idealist

- Un "program logic" este o colecție de proprietăți presupuse (sub formă de formule logice) 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

```
oslo → windy
oslo → norway
norway → cold
cold ∧ windy → winterIsComing
oslo
```

**Exemplu de întrebare.** Este adevărat `winterIsComing`?

# Prolog

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

## Program:

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

## Intrebare:

```
?- winterIsComing.  
true
```

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

## Sintaxă: constante, variabile, termeni compuși

- **Atomii**: `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(termin,..., termen)`
  - atom-ul care denumește termenul se numește **functor**
  - numărul de argumente se numește **aritate**

**Exercițiu.** 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'

**Exercițiu.** 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

**Exemplu.** 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.

### Exemplu.

```
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

**Program**

**Fapte + Reguli**

# 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**.

## Exemple.

- 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  $\leftarrow$ .

**Exemplu.** `comedy(X) :- griffin(X).`

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

Virgula `,` este conjuncția  $\wedge$ .

**Exemplu.** `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 definiții fiind un sau logic.

### Exemplu.

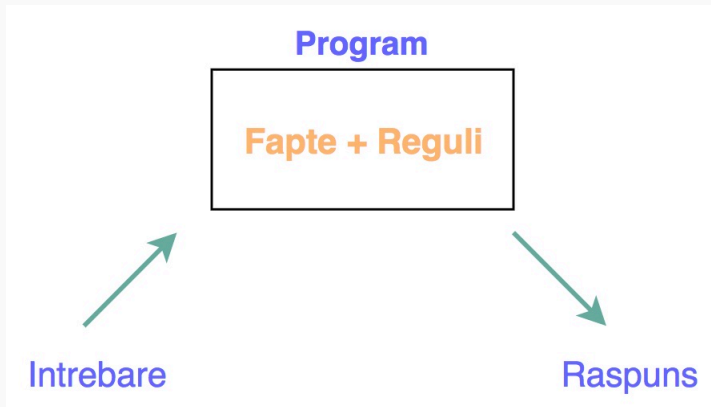
```
comedy(X) :- family_guy(X).  
comedy(X) :- south_park(X).  
comedy(X) :- disenchantment(X).
```

dacă family\_guy(X) este adevărat sau south\_park(X) este adevărat sau disenchantment(X) este adevărat,  
atunci comedy(X) este adevărat.

**Program**

**Fapte + Reguli**

Cum folosim un program 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:

**`?- predicat1(...), ..., predicatn(...).`**

- 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 răspunde la o întrebare se numește **țintă** (**goal**).

# Întrebări în Prolog

Prolog poate da 2 tipuri de răspunsuri:

- **false** – dacă întrebarea nu este o consecință a programului.
- **true** sau **valori pentru variabilele din întrebare** dacă întrebarea este o consecință a programului.

## Example

```
?- griffin(meg)
```

```
true
```

```
?- griffin(glenn)
```

```
false
```

```
?- griffin(X)
```

```
X = petr ;
```

```
X = lois ;
```

```
X = meg ;
```

```
X = stewie ;
```

```
false
```

## Cum găsește Prolog răspunsul

Pentru a găsi un răspuns,  
Prolog încearcă regulile în ordinea apariției lor.

**Exemplu.** Să presupunem că avem programul:

```
foo(a).  foo(b).  foo(c).
```

și că punem întrebarea:

```
?- foo(X).
```

```
X = a.
```

Pentru a răspunde la întrebare se caută o potrivire (unificator) între scopul `foo(X)` și baza de cunoștințe. Răspunsul este substituția care realizează unificarea, în cazul nostru `X = a`.

Răspunsul la întrebare este găsit prin unificare!

## Cum găsește Prolog răspunsul

**Exemplu.** Să presupunem că avem programul:

```
foo(a).  foo(b).  foo(c).
```

și că punem întrebările:

```
?- foo(X).
```

```
X = a.
```

```
?- foo(d).
```

```
false
```

Dacă nu se poate face unificarea, răspunsul este **false**.

## Cum găsește Prolog răspunsul

**Exemplu.** Să presupunem că avem programul:

```
foo(a).  foo(b).  foo(c).
```

și că punem întrebarea:

```
?- foo(X).
```

```
X = a.
```

Dacă dorim mai multe răspunsuri, tastăm ;

```
?- foo(X).
```

```
X = a ;
```

```
X = b ;
```

```
X = c.
```

# Cum găsește Prolog răspunsul

## Exemplu.

Să presupunem că avem programul:

foo(a).

foo(b).

foo(c).

și că punem întrebarea:

`?- foo(X).`

```
?- trace.  
true.  
  
[trace] ?- foo(X).  
  Call: (8) foo(_4556) ? creep  
  Exit: (8) foo(a) ? creep  
X = a ;  
  Redo: (8) foo(_4556) ? creep  
  Exit: (8) foo(b) ? creep  
X = b ;  
  Redo: (8) foo(_4556) ? creep  
  Exit: (8) foo(c) ? creep  
X = c.
```

# Cum găsește Prolog răspunsul

Pentru a găsi un răspuns, **Prolog redenumeste variabilele.**

## Exemplu.

Să presupunem că avem programul:

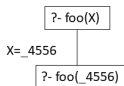
`foo(a).`

`foo(b).`

`foo(c).`

și că punem întrebarea:

`?- foo(X).`



# Cum găsește Prolog răspunsul

## Exemplu.

Să presupunem că avem programul:

`foo(a) .`

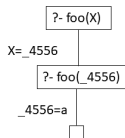
`foo(b) .`

`foo(c) .`

și că punem întrebarea:

`?- foo(X) .`

În acest moment, a fost găsită prima soluție: `X=_4556=a`.





# Cum găsește Prolog răspunsul

## Exemplu.

Să presupunem că avem programul:

foo(a).

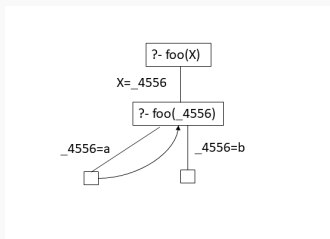
foo(b).

foo(c).

și că punem următoarea întrebare:

**?- foo(X).**

Dacă se dorește încă un răspuns, atunci se face un pas înapoi în **arborele de căutare** și se încearcă satisfacerea țintei cu o nouă valoare.



# Cum găsește Prolog răspunsul

## Exemplu.

Să presupunem că avem programul:

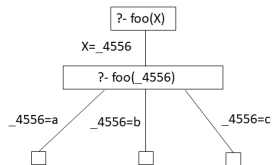
`foo(a).`

`foo(b).`

`foo(c).`

și că punem întrebarea:

`?- foo(X).`



arborele de căutare

# Cum găsește Prolog răspunsul

## Exemplu.

Să presupunem că avem programul:

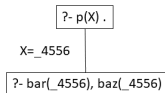
`bar(b) .`

`bar(c) .`

`baz(c) .`

și că punem întrebarea:

`?- bar(X), baz(X) .`



# Cum găsește Prolog răspunsul

Prolog se întoarce la ultima alegere dacă o subțintă eșuează.

## Exemplu.

Să presupunem că avem  
programul:

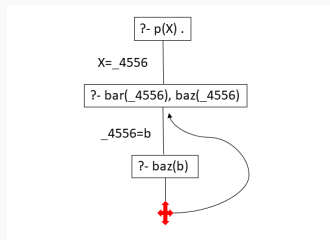
bar(b).

bar(c).

baz(c).

și că punem întrebarea:

?- bar(X), baz(X).



# Cum găsește Prolog răspunsul

## Exemplu.

Să presupunem că avem programul:

bar(b) .

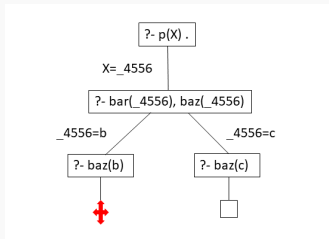
bar(c) .

baz(c) .

și că punem întrebarea:

?- bar(X) , baz(X) .

Soluția găsită este: X=\_4556=c.



## Cum găsește Prolog răspunsul

Ce se întâmplă dacă schimbăm ordinea regulilor?

### Exemplu.

Să presupunem că avem programul:

```
bar(c) .
```

```
bar(b) .
```

```
baz(c) .
```

și că punem întrebarea:

```
?- bar(X), baz(X) .
```

## Cum găsește Prolog răspunsul

Ce se întâmplă dacă schimbăm ordinea regulilor?

### Exemplu.

Să presupunem că avem programul:

```
bar(c) .
```

```
bar(b) .
```

```
baz(c) .
```

și că punem întrebarea:

```
?- bar(X), baz(X) .
```

```
X = c ;
```

```
false
```

Vă explicați ce s-a întâmplat? Desenați arborele de căutare!

## Problema colorării hărților

Să se coloreze o hartă dată cu o mulțime de culori dată astfel încât oricare două țări vecine să fie colorate diferit.

Cum modelăm această problemă în Prolog?



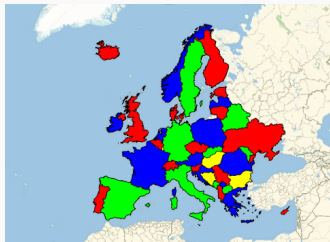
# Problema colorării hărților

Să se coloreze o hartă dată cu o mulțime de culori dată astfel încât oricare două țări vecine să fie colorate diferit.

Cum modelăm această problemă în Prolog?

Trebuie să definim:

- culorile
- harta
- constrângerile



Sursa imaginii

## Problema colorării hărților

Definim culorile, harta și constrângerile.

culoare(albastru).

culoare(rosu).

culoare(verde).

culoare(galben).

harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
vecin(RO,MD), vecin(RO,BG),  
vecin(RO,HU), vecin(UA,MD),  
vecin(BG,SE), vecin(SE,HU).

vecin(X,Y) :- culoare(X), culoare(Y), X \== Y.

## Problema colorării hărților

Definim culorile, harta și constrângerile. Cum punem întrebarea?

culoare(albastru).

culoare(rosu).

culoare(verde).

culoare(galben).

harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
vecin(RO,MD), vecin(RO,BG),  
vecin(RO,HU), vecin(UA,MD),  
vecin(BG,SE), vecin(SE,HU).

vecin(X,Y) :- culoare(X), culoare(Y), X \== Y.

## Problema colorării hărților

Definim culorile, harta și constrângerile. Cum punem întrebarea?

culoare(albastru).

culoare(rosu).

culoare(verde).

culoare(galben).

```
harta(RO,SE,MD,UA,BG,HU) :- vecin(RO,SE), vecin(RO,UA),  
                               vecin(RO,MD), vecin(RO,BG),  
                               vecin(RO,HU), vecin(UA,MD),  
                               vecin(BG,SE), vecin(SE,HU).
```

```
vecin(X,Y) :- culoare(X), culoare(Y), X \== Y.
```

```
?- harta(RO,SE,MD,UA,BG,HU).
```

Ce răspuns primim?

?- harta(RO,SE,MD,UA,BG,HU) .

RO = albastru,

SE = UA, UA = rosu,

MD = BG, BG = HU, HU = verde ■

## Liste în Prolog

- O listă în Prolog este un șir de elemente, separate prin virgulă, între paranteze drepte:

`[1,cold, parent(jon), [winter,is,coming],X]`

- O listă poate conține termeni de orice fel.
- Ordinea termenilor din listă are importanță:

`?- [1,2] == [2,1] .`

`false`

- Lista vidă se notează `[]`.
- Simbolul `|` desemnează coada listei:

`?- [1,2,3,4,5,6] = [X|T] .`

`X = 1, T = [2, 3, 4, 5, 6] .`

`?- [1,2,3|[4,5,6]] == [1,2,3,4,5,6] .`

`true.`

### Exerciții.

1. Definiți un predicat care verifică că un termen este lista.

## Exerciții.

1. Definiți un predicat care verifică că un termen este lista.

```
is_list([]).
```

```
is_list([_ | T]) :- is_list(T).
```



### Exerciții.

1. Definiți un predicat care verifică că un termen este lista.  
`is_list([]).`  
`is_list([_ | T]) :- is_list(T).`
2. Definiți predicate care verifică dacă un termen este primul element, ultimul element sau coada unei liste.

### Exerciții.

1. Definiți un predicat care verifică că un termen este lista.

```
is_list([]).
```

```
is_list([_ | T]) :- is_list(T).
```

2. Definiți predicate care verifică dacă un termen este primul element, ultimul element sau coada unei liste.

```
head([X|_],X).
```

```
last([X],X).
```

```
last([_|T],Y):- last(T,Y).
```

```
tail([],[]).
```

```
tail([_|T],T).
```

**Pe data viitoare!**

# Fundamentele limbajelor de programare

C10

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

# **Programare Logică**

## **Logica Clauzelor Horn**

---

## Program în Prolog = mulțime de predicate

Un exemplu de **program în Prolog** din cursul trecut:

```
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

## Spre logica din spatele Prologului

Pentru a putea modela universul programului, fixăm un alfabet/signatură/vocabular:

- o mulțime  $\mathbf{R}$  de simboluri de relații/predicate
- o mulțime  $\mathbf{F}$  de simboluri de operații/funcții
- o funcție aritate  $ar : \mathbf{F} \cup \mathbf{R} \rightarrow \mathbb{N}$
- Fie  $\mathbf{C} \subseteq \mathbf{F}$  mulțimea simbolurilor de operații de aritate 0, numite și simboluri de constante.

Notăm cu  $\mathbf{R}_n/\mathbf{F}_n$  mulțimea simbolurilor de relații/operații de aritate  $n$ .

Observație:  $\mathbf{F}_0 = \mathbf{C}$

### Exemplu.

- $\mathbf{R} = \mathbf{R}_1 \uplus \mathbf{R}_2$ , unde  $\mathbf{R}_1 = \{P\}$  și  $\mathbf{R}_2 = \{R\}$
- $\mathbf{F} = \mathbf{F}_0 \uplus \mathbf{F}_2$ , unde  $\mathbf{F}_0 = \mathbf{C} = \{c\}$  și  $\mathbf{F}_2 = \{f\}$

- Sintaxa Prolog nu face diferență între simboluri de operații și simboluri de predicate!
- Dar este important când ne uităm la teoria corespunzătoare programului în logică să facem această distincție.
- În sintaxa Prolog
  - termenii compuși sunt predicate: `father(peter, meg)`
  - operatorii sunt funcții: `+`, `*`, `mod`



Fixăm o mulțime numărabilă de **variabile**  $V$ .

Definim **termenii** inductiv astfel:

- orice variabilă este un termen
- orice simbol de constantă este un termen
- dacă  $f \in \mathbf{F}_n$ ,  $n > 0$  și  $t_1, \dots, t_n$  sunt termeni, atunci  $f(t_1, \dots, t_n)$  este termen.

**Exemple:**  $c$ ,  $x_1$ ,  $f(x_1, c)$ ,  $f(f(x_2, x_2), c)$

unde  $c \in \mathbf{C}$  reprezintă o constantă,  $x_1, x_2 \in V$  sunt variabile, iar  $f \in \mathbf{F}_2$  reprezintă un simbol de funcție de aritate 2.

Formulele atomice sunt definite astfel:

dacă  $R \in \mathbf{R}_0$ , atunci  $R$  este formulă atomică

dacă  $R \in \mathbf{R}_n$ ,  $n > 0$  și  $t_1, \dots, t_n$  sunt termeni,  
atunci  $R(t_1, \dots, t_n)$  este formulă atomică.

**Exemple:**  $P(f(x_1, c))$ ,  $R(c, x_2)$

unde  $c \in \mathbf{C}$ ,  $x_1, x_2 \in V$ ,  $f \in \mathbf{F}_2$ ,  $P \in \mathbf{R}_1$ , iar  $R \in \mathbf{R}_2$ .

# Formulele logicii de ordinul I

Definim **formulele** astfel:

- orice **formulă atomică** este o formulă
- dacă  $\varphi$  este o formulă, atunci  $\neg\varphi$  este o formulă
- dacă  $\varphi$  și  $\psi$  sunt formule, atunci  $\varphi \vee \psi$ ,  $\varphi \wedge \psi$ ,  $\varphi \rightarrow \psi$  sunt formule
- dacă  $\varphi$  este o formulă și  $x$  este o variabilă, atunci  $\forall x \varphi$ ,  $\exists x \varphi$  sunt formule

# Formulele logicii de ordinul I

Definim **formulele** astfel:

- orice **formulă atomică** este o formulă
- dacă  $\varphi$  este o formulă, atunci  $\neg\varphi$  este o formulă
- dacă  $\varphi$  și  $\psi$  sunt formule, atunci  $\varphi \vee \psi$ ,  $\varphi \wedge \psi$ ,  $\varphi \rightarrow \psi$  sunt formule
- dacă  $\varphi$  este o formulă și  $x$  este o variabilă, atunci  $\forall x \varphi$ ,  $\exists x \varphi$  sunt formule

**Exemplu:**

$$P(f(x_1, c)), \quad P(x_1) \vee P(c), \quad \forall x_1 P(x_1), \quad \forall x_2 R(x_2, x_1)$$

unde  $c \in \mathbf{C}$ ,  $x_1, x_2 \in V$ ,  $f \in \mathbf{F}_2$ ,  $P \in \mathbf{R}_1$ , iar  $R \in \mathbf{R}_2$ .

## Exercițiu

Fie alfabetul definit prin  $\mathbf{R} = \{<\}$ ,  $\mathbf{F} = \{s, +\}$ ,  $\mathbf{C} = \{0\}$  și  
 $ari(s) = 1$ ,  $ari(+)$  =  $ari(<) = 2$ .

Dați exemple de 3 termeni, 3 formule atomice și 3 formule.

## Exercițiu

Fie alfabetul definit prin  $\mathbf{R} = \{<\}$ ,  $\mathbf{F} = \{s, +\}$ ,  $\mathbf{C} = \{0\}$  și  
 $ari(s) = 1$ ,  $ari(+)$  și  $ari(<) = 2$ .

Dați exemple de 3 termeni, 3 formule atomice și 3 formule.

Exemple de **termeni**:

$0, x, s(0), s(s(0)), s(x), s(s(x)), \dots,$   
 $+(0, 0), +(s(s(0)), +(0, s(0))), +(x, s(0)), +(x, s(x)), \dots,$

Exemple de **formule atomice**:

$< (0, 0), < (x, 0), < (s(s(x)), s(0)), \dots$

Exemple de **formule**:

$\forall x \forall y < (x, +(x, y)), \forall x < (x, s(x))$

Un **literal** este o **formulă atomică** sau **negația** unei formule atomice.

O formulă este în **formă normală conjunctivă (FNC)** dacă este o **conjuncție** de **disjuncții** de **literal**i.

**Exemplu:**

$$(P(f(x_1, c)) \vee R(c, x_2)) \wedge \neg R(x_1, x_2) \wedge (R(x_1, x_1) \vee \neg P(c))$$

unde  $c \in \mathbf{C}$ ,  $x_1, x_2 \in V$ ,  $f \in \mathbf{F}_2$ ,  $P \in \mathbf{R}_1$ , iar  $R \in \mathbf{R}_2$ .

- O clauză este o disjuncție de literali.
- Dacă  $L_1, \dots, L_n$  sunt literali atunci vom reprezenta clauza  $L_1 \vee \dots \vee L_n$  ca mulțimea  $\{L_1, \dots, L_n\}$   
clauză = mulțime de literali
- O clauză  $C$  este trivială dacă conține un literal și complementul lui.
- Când  $n = 0$  obținem clauza vidă, care se notează  $\square$ .



Putem reprezenta o clauză prin mulțimea

$$\{\neg Q_1, \dots, \neg Q_n, P_1, \dots, P_k\}$$

unde  $n, k \geq 0$  și  $Q_1, \dots, Q_n, P_1, \dots, P_k$  sunt formule atomice.

Formula corespunzătoare este

$$\forall x_1 \dots \forall x_m (\neg Q_1 \vee \dots \vee \neg Q_n \vee P_1 \vee \dots \vee P_k)$$

unde  $x_1, \dots, x_m$  sunt toate variabilele care apar în clauză

Echivalent, putem scrie

$$\forall x_1 \dots \forall x_m (Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k)$$

Presupunem cuantificarea universală a clauzelor implicite:

$$Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k$$

## Clauze program definite

- Clauză  $Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k$   
unde  $n, k \geq 0$  și  $Q_1, \dots, Q_n, P_1, \dots, P_k$  sunt formule atomice.
- Dacă  $k = 1$ , atunci avem o clauză program definită:
  - cazul  $n > 0$  :  $Q_1 \wedge \dots \wedge Q_n \rightarrow P$
  - cazul  $n = 0$  :  $\top \rightarrow P$  (clauză unitate, fapt)  
 $\top$  este simbol pentru o formula mereu adevărată
- Program logic definit = mulțime finită de clauze definite

# Clauze Horn

- Clauză  $Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k$   
unde  $n, k \geq 0$  și  $Q_1, \dots, Q_n, P_1, \dots, P_k$  sunt formule atomice.
- Dacă  $k = 0$ , atunci avem o **clauză scop definită** (țintă, întrebare):

$$Q_1 \wedge \dots \wedge Q_n \rightarrow \perp$$

$\perp$  este simbol pentru o formula mereu falsă

Vom scrie o clauza scop definită ca  $Q_1, \dots, Q_n$ .

- În plus, dacă  $n = k = 0$ , atunci avem **clauza vidă**  $\square$

**Clauză Horn** = clauză program definită sau clauză scop ( $k \leq 1$ )

Limbajul **PROLOG** are la bază logica clauzelor Horn.

Clauza Horn = clauză program definită sau clauză scop ( $k \leq 1$ )

- Clauză  $Q_1 \wedge \dots \wedge Q_n \rightarrow P_1 \vee \dots \vee P_k$   
unde  $n, k \geq 0$  și  $Q_1, \dots, Q_n, P_1, \dots, P_k$  sunt formule atomice.
- Fie  $x_1, \dots, x_m$  toate variabilele care apar într-o clauza scop  $Q_1, \dots, Q_n$ . Atunci avem echivalența

$$\forall x_1 \dots \forall x_m (\neg Q_1 \vee \dots \vee \neg Q_n) \equiv \neg \exists x_1 \dots \exists x_m (Q_1 \wedge \dots \wedge Q_n)$$

Negația unei "întrebări" în PROLOG este clauză scop.

- **Logica clauzelor Horn:** un fragment al logicii de ordinul I în care singurele formule admise sunt **clauze Horn**
  - **formule atomice:**  $P(t_1, \dots, t_n)$
  - $Q_1 \wedge \dots \wedge Q_n \rightarrow P$   
unde toate  $Q_i, P$  sunt formule atomice,  $\top$  sau  $\perp$
- **Problema programării logice:** reprezentăm cunoștințele ca o mulțime de clauze definite  $KB$  și suntem interesați să aflăm răspunsul la o întrebare de forma  $Q_1 \wedge \dots \wedge Q_n$ , unde toate  $Q_i$  sunt formule atomice

$$KB \models Q_1 \wedge \dots \wedge Q_n$$

- Variabilele din  $KB$  sunt **cuantificate universal**.
- Variabilele din  $Q_1, \dots, Q_n$  sunt **cuantificate existențial**.

## Un exemplu

Fie următoarele clauze definite:

*father(jon, ken).*

*father(ken, liz).*

*father(X, Y)  $\rightarrow$  ancestor(X, Y)*

*daughter(X, Y)  $\rightarrow$  ancestor(Y, X)*

*ancestor(X, Y)  $\wedge$  ancestor(Y, Z)  $\rightarrow$  ancestor(X, Z)*

Putem pune întrebările:

- *ancestor(jon, liz)?*
- *ancestor(ken, Z)?*  
(există Z astfel încât *ancestor(ken, Z)*)

# Sistem de deducție pentru logica clauzelor Horn

Pentru un program logic definit  $KB$  avem

- **Axiome:** orice clauză din  $KB$
- **Regula de deducție *backchain*:**

$$\frac{\theta(Q_1) \quad \theta(Q_2) \quad \dots \quad \theta(Q_n) \quad (Q_1 \wedge Q_2 \wedge \dots \wedge Q_n \rightarrow P)}{\theta(Q)}$$

unde  $Q_1 \wedge Q_2 \wedge \dots \wedge Q_n \rightarrow P \in KB$ , iar  $\theta$  cmgu pentru  $Q$  și  $P$ .

Regula *backchain* conduce la un **sistem de deducție complet**:

Pentru o mulțime de clauze  $KB$  și o țintă  $Q$ ,  
dacă  $KB \models Q$ ,  
atunci există o derivare a lui  $Q$  folosind regula *backchain*.

## Cum răspundem la întrebări

Pentru o țintă  $Q$ , trebuie să găsim o clauză din  $KB$

$$Q_1 \wedge \dots \wedge Q_n \rightarrow P,$$

și un unificator  $\theta$  pentru  $Q$  și  $P$ .

În continuare vom verifica  $\theta(Q_1), \dots, \theta(Q_n)$ .

**Exemplu.** Pentru ținta

$$\text{ancestor}(\text{ken}, Z),$$

putem folosi clauză

$$\text{father}(X, Y) \rightarrow \text{ancestor}(X, Y)$$

cu unificatorul

$$\{X \mapsto \text{ken}, Y \mapsto Z\}$$

pentru a obține o nouă țintă

$$\text{father}(\text{ken}, Z).$$



## Rergula backchain

$$\frac{\theta(Q_1) \quad \theta(Q_2) \quad \dots \quad \theta(Q_n) \quad (Q_1 \wedge Q_2 \wedge \dots \wedge Q_n \rightarrow P)}{\theta(Q)}$$

unde  $Q_1 \wedge Q_2 \wedge \dots \wedge Q_n \rightarrow P \in KB$ , iar  $\theta$  este cgu pentru  $Q$  și  $P$ .

**Exemplu.** Presupunem că în KB avem:

*father(ken, liz).*

*father(X, Y) → ancestor(X, Y)*

$$\frac{\frac{father(ken, liz)}{father(ken, Z)} \quad father(X, Y) \rightarrow ancestor(X, Y)}{ancestor(ken, Z)}$$

Având doar această regulă, care sunt punctele de decizie în căutare?

- Ce clauză să alegem.
  - Pot fi mai multe clauze a căror parte dreaptă se potrivește cu o țintă.
  - Aceasta este o alegere de tip **SAU**: este suficient ca oricare din variante să reușească.
- Ordinea în care rezolvăm noile ținte.
  - Aceasta este o alegere de tip **ȘI**: trebuie arătate toate țintele noi.
  - Ordinea în care le rezolvăm poate afecta găsirea unei derivări, depinzând de strategia de căutare folosită.

Strategia de căutare din Prolog este de tip *depth-first*

- de sus în jos
  - pentru alegerile de tip **SAU**
  - alege clauzele în ordinea în care apar în program
- de la stânga la dreapta
  - pentru alegerile de tip **ȘI**
  - alege noile ținte în ordinea în care apar în clauza aleasă

## Regula backchain și rezoluția SLD

- Regula *backchain* este implementată în programarea logică prin rezoluția SLD (Selected, Linear, Definite).
- Prolog are la bază rezoluția SLD.

Fie  $KB$  o mulțime de clauze definite.

$$\text{SLD} \quad \boxed{\frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)}}$$

unde

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  este o clauză definită din  $KB$
- în care toate variabilele au fost redenumite cu variabile noi
- $\theta$  este cmgu pentru  $Q_i$  și  $Q$

## Rezoluția SLD - exemplu

```
father(eddard, sansa).  
father(eddard, jonSnow).
```

```
stark(eddard).           ?- stark(jonSnow)
stark(catelyn).
```

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

$$\text{SLD} \quad \frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)}$$

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  este o clauză definită din  $KB$
- variabilele din  $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  și  $Q_i$  se redenumesc
- $\theta$  este cmgu pentru  $Q_i$  și  $Q$ .

# Rezoluția SLD - exemplu

*father(eddard, sansa)*  
*father(eddard, jonSnow)*

$\neg \text{stark}(\text{jonSnow})$

---

*stark(eddard)*  
*stark(catelyn)*

$\theta(X) = \text{jonSnow}$

$\text{stark}(X) \vee \neg \text{father}(Y, X) \vee \neg \text{stark}(Y)$

$$\text{SLD} \left[ \frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)} \right]$$

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  este o clauză definită din *KB*
- variabilele din  $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  și  $Q_i$  se redenumesc
- $\theta$  este cmgu pentru  $Q_i$  și  $Q$ .

## Rezoluția SLD - exemplu

*father(eddard, sansa)*  
*father(eddard, jonSnow)*

*stark(eddard)*  
*stark(catelyn)*

*stark(X)  $\vee$   $\neg$ father(Y, X)  $\vee$   $\neg$ stark(Y)*

$$\frac{\neg \text{stark}(\text{jonSnow})}{\neg \text{father}(Y, \text{jonSnow}) \vee \neg \text{stark}(Y)}$$

$$\theta(X) = \text{jonSnow}$$

$$\text{SLD} \left[ \frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)} \right]$$

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  este o clauză definită din *KB*
- variabilele din  $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  și  $Q_i$  se redenumesc
- $\theta$  este cmgu pentru  $Q_i$  și  $Q$ .



# Rezoluția SLD - exemplu

*father(eddard, sansa)*  
*father(eddard, jonSnow)*

$$\frac{\neg \text{stark}(\text{jonSnow})}{\neg \text{father}(Y, \text{jonSnow}) \vee \neg \text{stark}(Y)}$$

*stark(eddard)*  
*stark(catelyn)*

$$\frac{\neg \text{father}(Y, \text{jonSnow}) \vee \neg \text{stark}(Y)}{\neg \text{stark}(\text{eddard})}$$

*stark(X)  $\vee$   $\neg$ father(Y, X)  $\vee$   $\neg$ stark(Y)*

$$\frac{\neg \text{stark}(\text{eddard})}{\square}$$

$$\text{SLD} \left[ \frac{\neg Q_1 \vee \dots \vee \neg Q_i \vee \dots \vee \neg Q_n}{\theta(\neg Q_1 \vee \dots \vee \neg P_1 \vee \dots \vee \neg P_m \vee \dots \vee \neg Q_n)} \right]$$

- $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  este o clauză definită din *KB*
- variabilele din  $Q \vee \neg P_1 \vee \dots \vee \neg P_m$  și  $Q_i$  se redenumesc
- $\theta$  este cmgu pentru  $Q_i$  și  $Q$ .

## Rezoluția SLD

Fie  $KB$  o mulțime de clauze definite și  $Q_1 \wedge \dots \wedge Q_m$  o întrebare, unde  $Q_i$  sunt formule atomice.

- O **derivare** din  $KB$  prin rezoluție SLD este o secvență

$$G_0 := \neg Q_1 \vee \dots \vee \neg Q_m, \quad G_1, \quad \dots, \quad G_k, \dots$$

în care  $G_{i+1}$  se obține din  $G_i$  prin regula **SLD**.

- Dacă există un  $k$  cu  $G_k = \square$  (clauza vidă), atunci derivarea se numește **SLD-respingere**.

# Rezoluția SLD

Fie  $KB$  o mulțime de clauze definite și  $Q_1 \wedge \dots \wedge Q_m$  o întrebare, unde  $Q_i$  sunt formule atomice.

- O **derivare** din  $KB$  prin rezoluție SLD este o secvență

$$G_0 := \neg Q_1 \vee \dots \vee \neg Q_m, \quad G_1, \quad \dots, \quad G_k, \dots$$

în care  $G_{i+1}$  se obține din  $G_i$  prin regula **SLD**.

- Dacă există un  $k$  cu  $G_k = \square$  (clauza vidă), atunci derivarea se numește **SLD-respingere**.

**Teoremă.** Sunt echivalente:

1. există o **SLD-respingere** a lui  $Q_1 \wedge \dots \wedge Q_m$  din  $KB$ ,
2.  $KB \models Q_1 \wedge \dots \wedge Q_m$ .

## Arbori SLD

- Presupunem că avem o mulțime de clauze definite  $KB$  și o țintă  $G_0 = \neg Q_1 \vee \dots \vee \neg Q_m$
- Construim un arbore de căutare (**arbore SLD**) astfel:
  - Fiecare nod al arborelui este o țintă (posibil vidă)
  - Rădăcina este  $G_0$
  - Dacă arborele are un nod  $G_i$ , iar  $G_{i+1}$  se obține din  $G_i$  folosind regula SLD folosind o clauză  $C_i \in KB$ , atunci nodul  $G_i$  are copilul  $G_{i+1}$ .  
Muchia dintre  $G_i$  și  $G_{i+1}$  este etichetată cu  $C_i$ .
- Dacă un arbore SLD cu rădăcina  $G_0$  are o frunză  $\square$  (clauza vidă), atunci există o SLD-respingere a lui  $G_0$  din  $KB$ .

### Exemplu.

Fie *KB* următoarea mulțime de clauze definite:

1. *grandfather*(*X*, *Z*) :  $\neg$ *father*(*X*, *Y*), *parent*(*Y*, *Z*)
2. *parent*(*X*, *Y*) :  $\neg$ *father*(*X*, *Y*)
3. *parent*(*X*, *Y*) :  $\neg$ *mother*(*X*, *Y*)
4. *father*(*ken*, *diana*)
5. *mother*(*diana*, *brian*)

Găsiți o respingere din *KB* pentru

?  $\neg$  *grandfather*(*ken*, *Y*)

### Exemplu.

Fie  $KB$  următoarea mulțime de clauze definite:

1.  $grandfather(X, Z) \vee \neg father(X, Y) \vee \neg parent(Y, Z)$
2.  $parent(X, Y) \vee \neg father(X, Y)$
3.  $parent(X, Y) \vee \neg mother(X, Y)$
4.  $father(ken, diana)$
5.  $mother(diana, brian)$

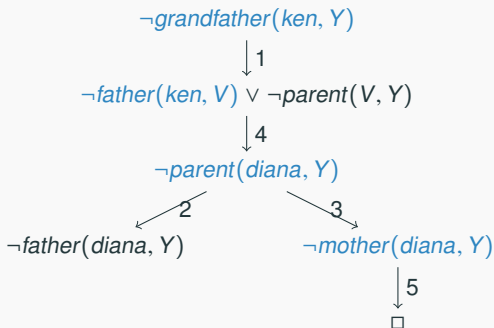
Găsiți o respingere din  $KB$  pentru

$$\neg grandfather(ken, Y)$$

# Rezoluția SLD - arbori de căutare

## Exemplu.

1.  $\text{grandfather}(X, Z) \vee \neg \text{father}(X, Y) \vee \neg \text{parent}(Y, Z)$
2.  $\text{parent}(X, Y) \vee \neg \text{father}(X, Y)$
3.  $\text{parent}(X, Y) \vee \neg \text{mother}(X, Y)$
4.  $\text{father}(\text{ken}, \text{diana})$
5.  $\text{mother}(\text{diana}, \text{brian})$



- Am arătat că **sistemul de inferență din spatele Prolog-ului este complet**.
  - Dacă o întrebare este consecință logică a unei mulțimi de clauze, atunci există o derivare a întrebării.
- Totuși, **strategia de căutate din Prolog este incompletă!**
  - Chiar dacă o întrebare este consecință logică a unei mulțimi de clauze, Prolog nu găsește mereu o derivare a întrebării.



```
warmerClimate :- albedoDecrease.  
warmerClimate :- carbonIncrease.  
iceMelts :- warmerClimate.  
albedoDecrease :- iceMelts.  
carbonIncrease.
```

```
?- iceMelts.
```

```
! Out of local stack
```

```
warmerClimate :- albedoDecrease.  
warmerClimate :- carbonIncrease.  
iceMelts :- warmerClimate.  
albedoDecrease :- iceMelts.  
carbonIncrease.
```

```
?- iceMelts.
```

```
! Out of local stack
```

## Limbaajul Prolog - exemplu

Totuși, există o derivare a lui *iceMelts* în sistemul de deducție din clauzele:

*albedoDecrease* → *warmerClimate*  
*carbonIncrease* → *warmerClimate*  
*warmerClimate* → *iceMelts*  
*iceMelts* → *albedoDecrease*  
⊥ → *carbonIncrease*

<i>carbonInc.</i>	<i>carbonInc. → warmerClim.</i>
<hr/>	
<i>warmerClim.</i>	
<i>warmerClim. → iceMelts</i>	
<hr/>	
<i>iceMelts</i>	

**Exercițiu** Desenați arborele SLD pentru programul Prolog de mai jos și ținta

?- p(X,X) .

1. p(X,Y) :- q(X,Z), r(Z,Y) .

2. p(X,X) :- s(X) .

3. q(X,b) .

4. q(b,a) .

5. q(X,a) :- r(a,X) .

6. r(b,a) .

7. s(X) :- t(X,a) .

8. s(X) :- t(X,b) .

9. s(X) :- t(X,X) .

10. t(a,b) .

11. t(b,a) .

# Rezoluția SLD - arbori de căutare

1.  $p(X, Y) :- q(X, Z), r(Z, Y).$

2.  $p(X, X) :- s(X).$

3.  $q(X, b).$

4.  $q(b, a).$

5.  $q(X, a) :- r(a, X).$

6.  $r(b, a).$

7.  $s(X) :- t(X, a).$

8.  $s(X) :- t(X, b).$

9.  $s(X) :- t(X, X).$

10.  $t(a, b).$

11.  $t(b, a).$

$p(X, Y) \vee \neg q(X, Z) \vee \neg r(Z, Y)$

$p(X, X) \vee \neg s(X)$

$q(X, b)$

$q(b, a)$

$q(X, a) \vee \neg r(a, X)$

$r(b, a)$

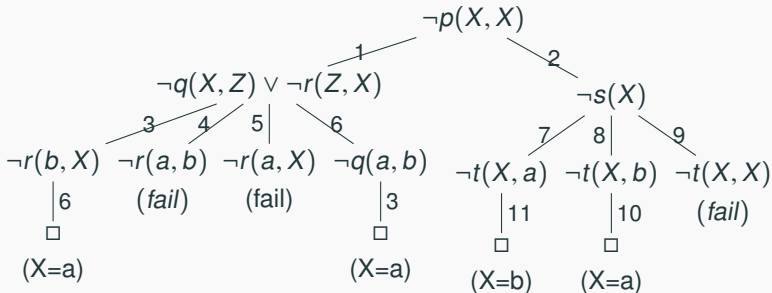
$s(X) \vee \neg t(X, a)$

$s(X) \vee \neg t(X, b)$

$s(X) \vee \neg t(X, X)$

$t(a, b)$

$t(b, a)$



**Pe data viitoare!**

# Fundamentele limbajelor de programare

C11

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

# **Semantica limbajelor de programare**

---



# Principalele paradigme de programare

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

## Fundamentele paradigmelor de programare

**Imperativă** Execuția unei Mașini Turing

**Logică** Rezoluția în logica clauzelor Horn

**Funcțională** Beta-reducție în Lambda Calcul

# Ce înseamnă semantică formală?

## Ce definește un limbaj de programare?

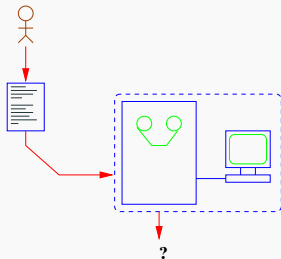
- **Sintaxa** – Simboluri de operație, cuvinte cheie, descriere (formală) a programelor/expresiilor bine formate
- **Practic** – Un limbaj e definit de modul cum poate fi folosit
  - Manual de utilizare și exemple de bune practici
  - Implementare (compilator/interpretor)
  - Instrumente ajutătoare (analizor de sintaxă, 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
  - 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
- 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 dacă 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".



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

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

- Este corect?

```
#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?

```
#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);  
}
```

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

**Conform standardului limbajului C (ISO/IEC 9899:2018)**

Comportamentul programului este nedefinit.

# Tipuri de semantică

- **Limbaj natural** – descriere textuală a efectelor
- **Statică** – un sistem de tipuri care exclude programe eronate
- **Operațională** – asocierea unei demonstrații pentru execuție
  - $\langle cod, \sigma \rangle \rightarrow \langle cod', \sigma' \rangle$
  - modelează execuția unui program pe o mașină abstractă
  - utilă pentru implementarea de compilatoare și interpretoare
- **Axiomatică** – aproximarea logică a efectelor unei instrucțiuni
  - $\vdash \{\varphi\}cod\{\psi\}$
  - modelează comportamentul un program prin formulele logice pe care le satisface
  - utilă pentru demonstrarea corectitudinii
- **Denotațională** – asocierea unui obiect matematic (denotație)
  - $\llbracket cod \rrbracket$
  - modelează un program ca obiecte matematice
  - utilă pentru fundamente matematice

Vom folosi ca exemplu un mic limbaj imperativ IMP care conține:

- **Expresii**
  - **Aritmetice:** `x + 3`
  - **Booleene:** `x >= 7`
- **Instrucțiuni**
  - **De atribuire:** `x = 5`
  - **Condiționale:** `if(x >= 7, x = 5, x = 0)`
  - **De ciclare:** `while(x >= 7, x = x - 1)`
- **Compunerea instrucțiunilor:** `x=7; while(x>=0, x=x-1)`
- **Blocuri de instrucțiuni:** `{x=7; while(x>=0, x=x-1)}`

Un exemplu de program în limbajul IMP

```
{ x = 10 ; sum = 0;  
  while(0 =< x,  
    {sum = sum + x; x = x-1}  
  )},  
sum
```

**Semantica:** după executia programului, se evaluează sum

## Sintaxa BNF a limbajului IMP

$E ::= n \mid x$   
 $\mid E + E \mid E - E \mid E * E$

$B ::= \text{true} \mid \text{false}$   
 $\mid E < E \mid E > E \mid E == E$   
 $\mid \text{not}(B) \mid \text{and}(B, B) \mid \text{or}(B, B)$

$C ::= \text{skip}$   
 $\mid x = E$   
 $\mid \text{if}(B, C, C)$   
 $\mid \text{while}(B, C)$   
 $\mid \{ C \} \mid C ; C$

$P ::= \{ C \}, E$

## **Semantica operațională small-step**

---

- **Semantica operațională** descrie cum se execută un program pe o mașină abstractă (ideală).
- **Semantica operațională small-step**
  - semantica structurală, a pașilor mici
  - descrie cum o execuție a programului avansează în funcție de reduceri succesive.

$$\langle cod, \sigma \rangle \rightarrow \langle cod', \sigma' \rangle$$

- **Semantica operațională big-step**
  - semantică naturală, într-un pas mare



# Starea execuției

- **Starea execuției** unui program IMP la un moment dat este dată de valorile deținute în acel moment de variabilele declarate în program.
- Formal, starea execuției unui program IMP la un moment dat este o **funcție parțială** (cu domeniu finit):

$$\sigma : Var \rightarrow Int$$

- **Notatii:**
  - Descrierea funcției prin enumerare:  $\sigma = n \mapsto 10, sum \mapsto 0$
  - Funcția vidă  $\perp$ , nedefinită pentru nicio variabilă
  - Obținerea valorii unei variabile:  $\sigma(x)$
  - Suprascrierea valorii unei variabile:

$$\sigma_{x \leftarrow v}(y) = \begin{cases} \sigma(y), & \text{dacă } y \neq x \\ v, & \text{dacă } y = x \end{cases}$$

# Semantica small-step

- Introdusă de Gordon Plotkin (1981)
- Denumiri alternative:
  - Semantică Operațională Structurală
  - semantică prin tranziții
  - semantică prin reducere
- Definește cel mai mic pas de execuție ca o relație „de tranziție” între configurații:

$$\langle cod, \sigma \rangle \rightarrow \langle cod', \sigma' \rangle$$

- Execuția se obține ca o succesiune de astfel de tranziții:

$$\begin{aligned}\langle x = 0 ; x = x + 1, \perp \rangle &\rightarrow \langle x = x + 1, x \mapsto 0 \rangle \\ &\rightarrow \langle x = 0 + 1, x \mapsto 0 \rangle \\ &\rightarrow \langle x = 1, x \mapsto 0 \rangle \\ &\rightarrow \langle \{\} , x \mapsto 1 \rangle\end{aligned}$$

- Cum definim această relație?

Prin inducție după elementele din sintaxă.

## Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

## Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)
  - Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

## Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

- Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

## Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

**if** (0 <= 5 + 7 \* **x** , r = 1 , r = 0)

- Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

$$\frac{\langle b , \sigma \rangle \rightarrow \langle b' , \sigma \rangle}{\langle \mathbf{if} (b, bl_1, bl_2) , \sigma \rangle \rightarrow \langle \mathbf{if} (b', bl_1, bl_2) , \sigma \rangle}$$

# Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

- Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

$$\frac{\langle b , \sigma \rangle \rightarrow \langle b' , \sigma \rangle}{\langle \text{if } (b, bl_1, bl_2) , \sigma \rangle \rightarrow \langle \text{if } (b', bl_1, bl_2) , \sigma \rangle}$$

- Axiome

- Realizează pasul computațional

# Redex. Reguli structurale. Axiome

- Expresie reductibilă (redex)

- Fragmentul de sintaxă care va fi procesat la pasul următor

`if (0 <= 5 + 7 * x , r = 1 , r = 0)`

- Reguli structurale

- Folosesc la identificarea următorului redex
- Definite recursiv pe structura termenilor

$$\frac{\langle b , \sigma \rangle \rightarrow \langle b' , \sigma \rangle}{\langle \text{if } (b, bl_1, bl_2) , \sigma \rangle \rightarrow \langle \text{if } (b', bl_1, bl_2) , \sigma \rangle}$$

- Axiome

- Realizează pasul computațional

$$\langle \text{if } (\text{true}, bl_1, bl_2) , \sigma \rangle \rightarrow \langle bl_1 , \sigma \rangle$$



# Semantica expresiilor aritmetice

- **Semantica unui întreg** este o valoare
  - nu poate fi redex, deci nu avem regulă

- **Semantica unei variabile**

$$(I_D) \quad \langle x, \sigma \rangle \rightarrow \langle i, \sigma \rangle \quad \text{dacă } \sigma(x) = i$$

- **Semantica adunării a două expresii aritmetice**

$$(ADD) \quad \langle i_1 + i_2, \sigma \rangle \rightarrow \langle i, \sigma \rangle \quad \text{dacă } i_1 + i_2 = i$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle}$$

$$\frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a_1 + a'_2, \sigma \rangle}$$

Observatie: ordinea de evaluare a argumentelor este nespecificată.

# Semantica expresiilor booleene

- Semantica operatorului de comparație

(LEQ-FALSE)  $\langle i_1 \leq i_2, \sigma \rangle \rightarrow \langle \mathbf{false}, \sigma \rangle$  dacă  $i_1 > i_2$

(LEQ-TRUE)  $\langle i_1 \leq i_2, \sigma \rangle \rightarrow \langle \mathbf{true}, \sigma \rangle$  dacă  $i_1 \leq i_2$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 \leq a_2, \sigma \rangle \rightarrow \langle a'_1 \leq a_2, \sigma \rangle} \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle a_1 \leq a_2, \sigma \rangle \rightarrow \langle a_1 \leq a'_2, \sigma \rangle}$$

- Semantica negației

(!-FALSE)  $\langle \mathbf{not(true)}, \sigma \rangle \rightarrow \langle \mathbf{false}, \sigma \rangle$

(!-TRUE)  $\langle \mathbf{not(false)}, \sigma \rangle \rightarrow \langle \mathbf{true}, \sigma \rangle$

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle \mathbf{not}(a), \sigma \rangle \rightarrow \langle \mathbf{not}(a'), \sigma \rangle}$$

- Semantica și-ului

(AND-FALSE)  $\langle \text{and}(\text{false}, b_2), \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle$

(AND-TRUE)  $\langle \text{and}(\text{true}, b_2), \sigma \rangle \rightarrow \langle b_2, \sigma \rangle$

$$\frac{\langle b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle}{\langle \text{and}(b_1, b_2), \sigma \rangle \rightarrow \langle \text{and}(b'_1, b_2), \sigma \rangle}$$

- Semantica blocurilor

$$(\text{BLOCK}) \quad \langle \{ s \}, \sigma \rangle \rightarrow \langle s, \sigma \rangle$$

- Semantica compunerii secvențiale

$$(\text{NEXT-STMT}) \quad \langle \mathbf{skip}; s_2, \sigma \rangle \rightarrow \langle s_2, \sigma \rangle$$

$$\frac{\langle s_1, \sigma \rangle \rightarrow \langle s'_1, \sigma' \rangle}{\langle s_1 ; s_2, \sigma \rangle \rightarrow \langle s'_1 ; s_2, \sigma' \rangle}$$

- Semantica atribuirii

$$(\text{ASGN}) \quad \langle x = i, \sigma \rangle \rightarrow \langle \mathbf{skip}, \sigma' \rangle \quad \text{dacă } \sigma' = \sigma_{x \leftarrow i}$$

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle x = a, \sigma \rangle \rightarrow \langle x = a', \sigma \rangle}$$

# Semantica lui if

- Semantica lui if

(IF-TRUE)  $\langle \mathbf{if}(\mathbf{true}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_1, \sigma \rangle$

(IF-FALSE)  $\langle \mathbf{if}(\mathbf{false}, bl_1, bl_2), \sigma \rangle \rightarrow \langle bl_2, \sigma \rangle$

$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle \mathbf{if}(b, bl_1, bl_2), \sigma \rangle \rightarrow \langle \mathbf{if}(b', bl_1, bl_2), \sigma \rangle}$$

- Semantica lui while

(WHILE)  $\langle \mathbf{while}(b, bl), \sigma \rangle \rightarrow \langle \mathbf{if}(b, bl; \mathbf{while}(b, bl), \mathbf{skip}), \sigma \rangle$

- Semantica programelor

(PGM)  $\frac{\langle a_1, \sigma_1 \rangle \rightarrow \langle a_2, \sigma_2 \rangle}{\langle (\mathbf{skip}, a_1), \sigma_1 \rangle \rightarrow \langle (\mathbf{skip}, a_2), \sigma_2 \rangle}$

$$\frac{\langle s_1, \sigma_1 \rangle \rightarrow \langle s_2, \sigma_2 \rangle}{\langle (s_1, a), \sigma_1 \rangle \rightarrow \langle (s_2, a), \sigma_2 \rangle}$$

Execuție pas cu pas

$\langle i = 3 ; \text{while } (0 \leq i , \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{ASGN}}$

## Execuție pas cu pas

$$\langle i = 3 ; \text{while } (0 \leq i , \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{ASGN}} \\ \langle \text{skip; while } (0 \leq i , \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{NEXT-STMT}}$$

## Execuție pas cu pas

$$\begin{aligned} \langle i = 3 ; \text{while } (0 \leq i , \{ i = i + -4 \}) , \perp \rangle &\xrightarrow{\text{ASGN}} \\ \langle \text{skip} ; \text{while } (0 \leq i , \{ i = i + -4 \}) , i \mapsto 3 \rangle &\xrightarrow{\text{NEXT-STMT}} \\ \langle \text{while } (0 \leq i , \{ i = i + -4 \}) , i \mapsto 3 \rangle &\xrightarrow{\text{WHILE}} \end{aligned}$$



# Semantica small-step a lui IMP

## Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{ASGN}} \\ &\langle \text{skip} ; \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{NEXT-STMT}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{WHILE}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{ID}} \end{aligned}$$

# Semantica small-step a lui IMP

## Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{ASGN}} \\ &\langle \text{skip} ; \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{NEXT-STMT}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{WHILE}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{ID}} \\ &\langle \text{if } (0 \leq 3, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{LEQ-TRUE}} \end{aligned}$$

# Semantica small-step a lui IMP

## Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{ASGN}} \\ &\langle \text{skip} ; \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{NEXT-STMT}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{WHILE}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{ID}} \\ &\langle \text{if } (0 \leq 3, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{LEQ-TRUE}} \\ &\langle \text{if } (\text{true}, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{IF-TRUE}} \end{aligned}$$

# Semantica small-step a lui IMP

## Execuție pas cu pas

$$\begin{aligned} &\langle i = 3 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \perp \rangle \xrightarrow{\text{ASGN}} \\ &\langle \text{skip} ; \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{NEXT-STMT}} \\ &\langle \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{WHILE}} \\ &\langle \text{if } (0 \leq i, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{ID}} \\ &\langle \text{if } (0 \leq 3, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{LEQ-TRUE}} \\ &\langle \text{if } (\text{true}, i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , \text{skip}) , i \mapsto 3 \rangle \xrightarrow{\text{IF-TRUE}} \\ &\langle i = i + -4 ; \text{while } (0 \leq i, \{ i = i + -4 \}) , i \mapsto 3 \rangle \xrightarrow{\text{ID}} \\ &\dots \end{aligned}$$

## **Semantica axiomatică**

---

- Dezvoltată de Tony Hoare în 1969 (inspirată de rezultatele lui Robert Floyd).
- Definește triplete (**triplete Hoare**) de forma

$$\{P\} \mathbb{C} \{Q\}$$

unde:

- $\mathbb{C}$  este o instrucțiune
- $P$  (precondiție),  $Q$  (postcondiție) sunt aserțiuni logice asupra stării sistemului înaintea, respectiv după execuția lui  $\mathbb{C}$
- Limbajul aserțiunilor este un limbaj de ordinul I.

## Interpretarea unui triplet Hoare $\{P\} \mathbb{C} \{Q\}$

- dacă programul se execută dintr-o stare inițială care satisface  $P$
- și execuția se termină
- atunci se ajunge într-o stare finală care satisface  $Q$ .

## Interpretarea unui triplet Hoare $\{P\} \mathbb{C} \{Q\}$

- dacă programul se execută dintr-o stare inițială care satisface  $P$
- și execuția se termină
- atunci se ajunge într-o stare finală care satisface  $Q$ .

## Exemple:

- $\{x = 1\} x = x+1 \{x = 2\}$  este corect
- $\{x = 1\} x = x+1 \{x = 3\}$  **nu** este corect
- $\{\top\} \text{if } (x \leq y) \ z = x; \text{ else } z = y; \{z = \min(x, y)\}$  este corect



## Logica Hoare ne ajută să verificăm **corectitudinea** programelor

- Se asociază fiecărei construcții sintactice o regulă de deducție care definește recursiv tripletele corecte pentru un limbaj.
- Se exprimă o aserțiune de corectitudine a programului ca un triplet Hoare
- Se verifică dacă tripletul dat e corect folosind definiția recursivă

## Reguli generale pentru logică propozițională

$$(\rightarrow) \frac{P1 \rightarrow P2 \quad \{P2\} \mathbb{C} \{Q2\} \quad Q2 \rightarrow Q1}{\{P1\} \mathbb{C} \{Q1\}}$$

$$(\vee) \frac{\{P1\} \mathbb{C} \{Q\} \quad \{P2\} \mathbb{C} \{Q\}}{\{P1 \vee P2\} \mathbb{C} \{Q\}}$$

$$(\wedge) \frac{\{P\} \mathbb{C} \{Q1\} \quad \{P\} \mathbb{C} \{Q2\}}{\{P\} \mathbb{C} \{Q1 \wedge Q2\}}$$

$$(\text{SKIP}) \quad \overline{\{P\} \{\} \{P\}}$$

$$(\text{SEQ}) \quad \frac{\{P\} C_1 \{Q\} \quad \{Q\} C_2 \{R\}}{\{P\} C_1; C_2 \{R\}}$$

$$(\text{ASIGN}) \quad \overline{\{P[x/e]\} x = e \{P\}}$$

$$(\text{IF}) \quad \frac{\{B \wedge P\} C_1 \{Q\} \quad \{\neg B \wedge P\} C_2 \{Q\}}{\{P\} \text{if } (B) C_1 \text{ else } C_2 \{Q\}}$$

$$(\text{WHILE}) \quad \frac{\{B \wedge P\} C \{P\}}{\{P\} \text{while } (B) C \{\neg B \wedge P\}}$$

$$(A_{\text{SIGN}}) \quad \frac{}{\{P[x/e]\} \ x = e \ \{P\}}$$

**Exemplu:**

$$\{x + y = y + 10\} \ x = x + y \ \{x = y + 10\}$$

$$(\text{If}) \quad \frac{\{B \wedge P\} C_1 \{Q\} \quad \{\neg B \wedge P\} C_2 \{Q\}}{\{P\} \text{ if } (B) C_1 \text{ else } C_2 \{Q\}}$$

### Exemplu:

Pentru a demonstra

$\{\top\} \text{ if } (x \leq y) \ z = x; \text{ else } z = y; \{z = \min(x, y)\}$

este suficient să demonstrăm

- $\{x \leq y\} \ z = x \ \{z = \min(x, y)\}$
- $\{\neg(x \leq y)\} \ z = y \ \{z = \min(x, y)\}$

## Invarianți pentru while

Cum demonstrăm  $\{P\} \text{ while}(B) \ C \ \{Q\}$  ?

Se determină un invariant  $I$  și se folosește următoarea regulă:

$$(\text{Inv}) \quad \frac{P \rightarrow I \quad \{B \wedge I\} \ C \ \{I\} \quad (I \wedge \neg B) \rightarrow Q}{\{P\} \ \mathbf{while} \ (B) \ C \ \{Q\}}$$

Invariantul trebuie să satisfacă următoarele proprietăți:

- să fie adevărat inițial
- să rămână adevărat după execuția unui ciclu
- să implice postcondiția la ieșirea din buclă

## Invarianti pentru while

$\{x = 0 \wedge 0 \leq n \wedge y = 1\}$

`while (x < n) { x = x + 1; y = y * x }`

$\{y = n!\}$

## Invarianti pentru while

$\{x = 0 \wedge 0 \leq n \wedge y = 1\}$

`while (x < n) { x = x + 1; y = y * x }`

$\{y = n!\}$

- Invariantul / este  $y = x!$



## Invarianți pentru while

$\{x = 0 \wedge 0 \leq n \wedge y = 1\}$

`while (x < n) { x = x + 1; y = y * x }`

$\{y = n!\}$

- Invariantul  $I$  este  $y = x!$
- $(x = 0 \wedge 0 \leq n \wedge y = 1) \rightarrow I$
- $\{I \wedge (x < n)\} \quad x = x + 1; y = y * x \quad \{I\}$
- $I \wedge \neg(x < n) \rightarrow (y = n!)$

**Pe data viitoare!**

# Fundamentele limbajelor de programare

C12

---

Denisa Diaconescu

Traian Șerbănuță

Departamentul de Informatică, FMI, UB

## **Imagine de ansamblu**

---

- Lambda calcul fără tipuri
  - Terminologie
  - $\beta$ -reducții
  - Strategii de evaluare
  - Expresivitate (codări)
  - Puncte fixe (combinatori de punct fix)

- Lambda calcul cu tipuri simple
  - Terminologie
  - Church-typing vs. Curry-typing
  - Sistem de deducție ( $\lambda \rightarrow$ ) pentru calculul Church
  - Alte tipuri (unit, void, produs, suma)
  - Corespondența Curry-Howard
  - Type checking/ Type inference
  - Sistem de deducție ( $\lambda \rightarrow$ ) cu constrângeri pentru calculul Church

# Ce am parcurs la acest curs

- Unificare
  - Terminologie
  - Algoritmul de unificare
- Programare logică
  - Terminologie
  - Prolog
  - Logica clauzelor Horn
  - Sistem de deducție pentru logica clauzelor Horn (regula backchain)
  - Strategia de căutare din Prolog
  - Rezoluția SLD (arbori de căutare)

# Ce am parcurs la acest curs

- Semantica limbajelor de programare
  - Terminologie
  - Semantica operațională small-step
  - Semantica axiomatică



# Examen

---

- 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~~  
un material pus la dispoziție de noi și **o foaie** cu ce notițe vreți voi
- **aparatele electronice nu sunt permise la examen**



**Mulțumim că ați participat la acest curs!**

**Baftă la examen și mai departe!**