Fundamentele limbajelor de programare

C01

Denisa Diaconescu
Traian Șerbănuță

Departamentul de Informatică, FMI, UB

Organizare

Instructori

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

Resurse

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

Prezență

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

Evaluare

Notare

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

Condiție de promovabilitate

- cel puțin 5 > 4.99

Nota laborator

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

Examen parțial

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

Examen final

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

Imagine de ansamblu asupra materiei

Curs

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

Laborator

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

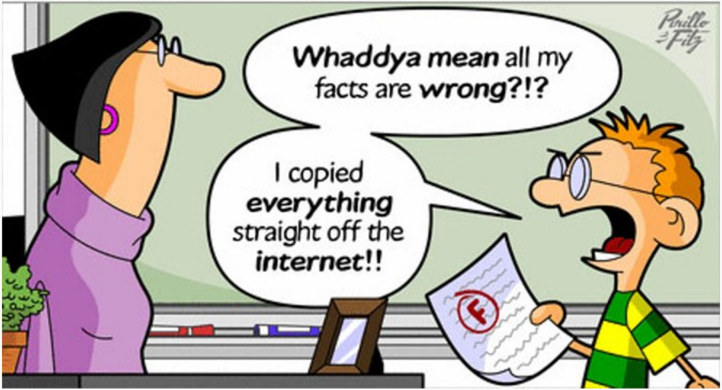
Bibliografie

- H. Barendregt, E. Barendsen, **Introduction to Lambda Calculus**, 2000.
- R. Nederpelt, H. Geuvers , **Type Theory and Formal Proof**. Cambridge University Press, 2014.
- B.C. Pierce, **Types and programming languages**. MIT Press, 2002
- P. Selinger,**Lecture Notes on the Lambda Calculus**. Dep. of Mathematics and Statistics, Dalhousie University, Canada.
- P. Blackburn, J. Bos, and K. Striegnitz, **Learn Prolog Now!** (Texts in Computing, Vol. 7),College Publications, 2006
- M. Huth, M. Ryan, **Logic in Computer Science (Modelling and Reasoning about Systems)**, Cambridge University Press, 2004.
- J. Lloyd. **Foundations of Logic Programming**, second edition. Springer, 1987.

La acest curs vom folosi destul de mult literele grecești

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

Nu trișați, cereți-ne ajutorul!



Ce și de ce lambda calcul?

Ce este o funcție în matematică?

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

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

Ce este o funcție în matematică?

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

Extensional vs. intensional

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

O paranteză: expresii aritmetice

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

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

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

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

  în loc de

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

Lambda calcul

- Lambda calculul este o teorie a funcțiilor ca formule.
- Este un sistem care permite manipularea funcțiilor ca expresii. Extindem intuiția de la expresii aritmetice pentru funcții.
- De exemplu, dacă în mod normal am scrie

  $$\text{Fie } f \text{ funcția } x \mapsto x^2. \text{ Atunci } A = f(5),$$

  în lambda calcul scriem doar

  $$A = (\lambda x.x^2)(5).$$

- Expresia $\lambda x.x^2$ reprezintă funcția care duce x în x^2 (nu instrucțiunea/declarația că x este dus în x^2).
- Variabila x este locală/legată în termenul $\lambda x.x^2$ De aceea, nu contează dacă am fi scris $\lambda y.y^2$

Funcții de nivel înalt

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

Lambda calcul fără tipuri vs cu tipuri

Cateva exemple:

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

Lambda calcul fără tipuri vs cu tipuri

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

- Pentru funcția identitate $f = \lambda x.x$ avem $f(x) = x$, pentru orice x. În particular, putem lua $x = f$ si obținem
$$f(f) \simeq (\lambda x.x)(\lambda x.x) \simeq \lambda x.x \simeq f.$$
- Combinatorul $\omega = \lambda x.xx$ care pentru un x reprezintă funcția care aplică x lui x
$$\omega(\lambda y.y) \simeq (\lambda x.xx)(\lambda y.y) \simeq (\lambda y.y)(\lambda y.y) \simeq (\lambda y.y)$$
Ce reprezintă $\omega(\omega)$?

Lambda calcul fără tipuri vs cu tipuri

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

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

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

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

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

Ce reprezintă $\omega(\omega)$?

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

Lambda calcul

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

Calculabilitate

- Una din marile întrebări din anii 1930:

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

- O definiție informală:

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

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

Definiții pentru Calculabilitate

1. Turing – a definit un calculator ideal numit mașina Turing și a postulat că o funcție este calculabilă ddacă poate fi calculată de o astfel de mașină.

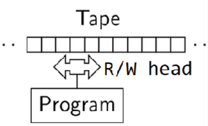
2. Gödel – a definit clasa funcțiilor recursive și a postulat că o funcție este calculabilă ddacă este o funcție recursivă.

3. Church – a definit un limbaj de programare ideal numit lambda calcul și a postulat că o funcție este calculabilă ddacă poate fi scrisă ca un lambda termen.

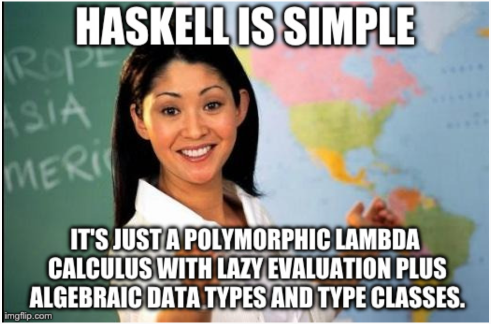
Teza Church-Turing

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

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



Lambda calcul în informatică



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

Proofs-as-programs

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

Quiz time!

https://tinyurl.com/C01-Quiz1

Pe săptămâna viitoare!

Fundamentele limbajelor de programare

C02

Denisa Diaconescu
Traian Șerbănuță

Departamentul de Informatică, FMI, UB

Lambda calcul - elemente de bază

Lambda calcul

- 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 termeni

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

Mulțimea lambda termenilor este dată de următoarea formă BNF:

$$\begin{array}{rcl} \text{lambda termen} & = & \text{variabilă} \\ & \mid & \text{aplicare} \\ & \mid & \text{abstractizare} \end{array}$$

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

Example

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

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 λ 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 $x, y, z, \ldots$.

Fie A un alfabet format din elementele din V, și simbolurile speciale "(", ")", "λ" si ".".

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
  - $M\,N\,P$ înseamnă $(M\,N)\,P$
  - $f\,x\,y\,z$ înseamnă $((f\,x)\,y)\,z$
- Corpul abstractizării (partea de după punct) se extinde la dreapta cât se poate
  - $\lambda x.M\,N$ înseamnă $\lambda x.(M\,N)$, nu $(\lambda x.M)\,N$
- Mai mulți λ pot fi comprimați
  - $\lambda xyz.M$ este o abreviere pentru $\lambda x.\lambda y.\lambda z.M$

Aceste convenții nu afectează definiția lambda termenilor.

Exerciții

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

1. $(\lambda x.(\lambda y.(\lambda z.((x\, z)(y\, z)))))$
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 variable 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:

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

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.

$$
\begin{aligned}
x\langle y/x\rangle &\equiv y, & \\
z\langle y/x\rangle &\equiv z, & \text{dacă } x \neq z \\
(M\,N)\langle y/x\rangle &\equiv (M\langle y/x\rangle)\,(N\langle y/x\rangle) & \\
(\lambda x.M)\langle y/x\rangle &\equiv \lambda y.(M\langle y/x\rangle) & \\
(\lambda z.M)\langle y/x\rangle &\equiv \lambda z.(M\langle y/x\rangle), & \text{dacă } x \neq z
\end{aligned}
$$

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 α-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

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

α-echivalență

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

$$(\mathit{refl})\quad \frac{}{M = M} \qquad\qquad (\mathit{cong})\quad \frac{M = M' \qquad N = N'}{MN = M'N'}$$

$$(\mathit{symm})\quad \frac{M = N}{N = M} \qquad\qquad (\xi)\quad \frac{M = M'}{\lambda x.M = \lambda x.M'}$$

$$(\mathit{trans})\quad \frac{M = N \qquad N = P}{M = P} \qquad\qquad (\alpha)\quad \frac{y \notin M}{\lambda x.M = \lambda y.(M\{y/x\})}$$

Convenția Barendregt:

variabilele legate sunt redenumite pentru a fi distincte.

Substituții

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 $N\,(\lambda Ny.N)$.

Substituții

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:

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

Deaorece nu specificăm ce variabilă nouă alegem,
spunem că substituția este bine-definită modulo α-echivalențe.

Exerciții

Exercițiu. Calculați următoarele substituții:

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

Exerciții

Exercițiu. Calculați următoarele substituții:

1. $(\lambda z.x)[y/x]$ Corect: $\lambda z.y$
2. $(\lambda y.x)[y/x]$
3. $(\lambda y.x)[(\lambda z.z\, w)/x]$

Exerciții

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

Exerciții

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 - β-reducții

β-reducții

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

- β-reducție = procesul de a evalua lambda termeni prin "pasarea de argumente funcțiilor"
- β-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 acelui 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

β-reducții

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

$$(\beta) \qquad \frac{}{(\lambda x.M)N \to_\beta M[N/x]}$$

$$(cong_1) \qquad \frac{M \to_\beta M'}{MN \to_\beta M'N}$$

$$(cong_2) \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

$$(\xi) \qquad \frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'}$$

β-reducții

La fiecare pas, subliniem redexul ales în procesul de β-reducție.

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

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

β-reducții

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

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

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

β-formă normală

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

$$\begin{aligned} \underline{(\lambda x.x\,x)\,(\lambda x.x\,x)} \quad &\rightarrow_\beta \quad (\lambda x.x\,x)\,(\lambda x.x\,x) \\ &\rightarrow_\beta \quad \ldots \end{aligned}$$

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

β-formă normală

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

$$\begin{aligned}
\underline{(\lambda xy.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x))}\,(\lambda z.z) &\to_\beta \underline{(\lambda y.y)\,(\lambda x.x)} \\
&\to_\beta \lambda x.x
\end{aligned}$$

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

Contează strategia de evaluare.

β-formă normală

Notăm cu $M \twoheadrightarrow_\beta M'$ faptul că M poate fi β-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 $M \twoheadrightarrow_\beta N$.

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 β-reducției

Teorema Church-Rosser. Dacă $M \twoheadrightarrow_\beta M_1$ și $M \twoheadrightarrow_\beta M_2$ atunci există M' astfel încât $M_1 \twoheadrightarrow_\beta M'$ și $M_2 \twoheadrightarrow_\beta M'$.

$$
\begin{tikzcd}
 & M \arrow[dl, two heads] \arrow[dr, two heads] & \\
M_1 \arrow[dr, two heads, dashed] & & M_2 \arrow[dl, two heads, dashed] \\
 & M' &
\end{tikzcd}
$$

Consecință. Un lambda termen are cel mult o β-formă normală (modulo α-echivalență).

Exerciții

Exercițiu. Verificați dacă termenii de mai jos pot fi aduși la o β-formă normală:

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

Exerciții

Exercițiu. Verificați dacă termenii de mai jos pot fi aduși la o β-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: $(\lambda y.y\, y\, y)\, (\lambda y.y\, y\, y)\, (\lambda y.y\, y\, y) \ldots$

Strategii de evaluare

Strategii de evaluare

De cele mai multe ori, există mai mulți pași de β-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 reducție.

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.

$$
\begin{aligned}
\underline{(\lambda xy.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x))}\,(\lambda z.z) &\to_\beta \underline{(\lambda y.y)\,(\lambda x.x)} \\
&\to_\beta \lambda x.x
\end{aligned}
$$

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.

$$(\lambda xy.y)\,\underline{((\lambda x.x\,x)\,(\lambda x.x\,x))}\,(\lambda z.z) \quad \to_\beta \quad (\lambda xy.y)\,((\lambda x.x\,x)\,(\lambda x.x\,x))\,(\lambda z.z)$$

Strategii în programare funcțională

În limbaje de programare funcțională, în general, reducerile din corpul unei λ-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 λ-abstractizări

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

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

CBN vs CBV

O valoare este un λ-term pentru care nu există β-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 β-reducția $(\lambda x.M)\ N \twoheadrightarrow_\beta M[N/x]$ 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.

CBN vs CBV

Example

Considerăm 3 și *succ* primitive.

Strategia CBV:

$$\begin{array}{rcl} (\lambda x.succ\, x)\,((\lambda y.succ\, y)\, 3) & \twoheadrightarrow_\beta & (\lambda x.succ\, x)\,(succ\, 3) \\ & \to & (\lambda x.succ\, x)\, 4 \\ & \twoheadrightarrow_\beta & succ\, 4 \\ & \to & 5 \end{array}$$

Strategia CBN:

$$\begin{array}{rcl} (\lambda x.succ\, x)\,((\lambda y.succ\, y)\, 3) & \twoheadrightarrow_\beta & succ\,((\lambda y.succ\, y)\, 3) \\ & \twoheadrightarrow_\beta & succ\,(succ\, 3) \\ & \to & succ\, 4 \\ & \to & 5 \end{array}$$

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 λ-calculului

Expresivitatea λ-calculului

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

Vom vedea cum putem reprezenta:

- valori booleene
- numere naturale

Booleeni

Booleeni

Vrem să definim λ-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)

Booleeni

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

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 λ-termeni oarecare.

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

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

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

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

Booleeni

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

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

$$\begin{aligned} \mathbf{and} &\triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ b_2\ \mathbf{F} \\ \mathbf{or} &\triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ \mathbf{T}\ b_2 \\ \mathbf{not} &\triangleq \lambda b_1.\mathbf{if}\ b_1\ \mathbf{F}\ \mathbf{T} \end{aligned}$$

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

Nu există nicio garanție să se comporte rezonabil pe orice alți λ-termeni.

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

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

Booleeni

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

$$\begin{aligned} \mathbf{and} &\triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ b_2\ \mathbf{F} \\ \mathbf{or} &\triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ \mathbf{T}\ b_2 \\ \mathbf{not} &\triangleq \lambda b_1.\mathbf{if}\ b_1\ \mathbf{F}\ \mathbf{T} \end{aligned}$$

Exercițiu. Aduceți la o formă normală următorii termenii:

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

Booleni

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

$$\begin{aligned} \mathbf{and} &\triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ b_2\ \mathbf{F} \\ \mathbf{or} &\triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ \mathbf{T}\ b_2 \\ \mathbf{not} &\triangleq \lambda b_1.\mathbf{if}\ b_1\ \mathbf{F}\,\mathbf{T} \end{aligned}$$

Soluții:

$$\begin{aligned} \mathbf{and}\,\mathbf{T}\mathbf{F} &= (\lambda b_1 b_2.\mathbf{if}\ b_1\ b_2\ \mathbf{F})\,\mathbf{T}\mathbf{F} \twoheadrightarrow_\beta \mathbf{if}\,\mathbf{T}\,\mathbf{F}\,\mathbf{F} = (\lambda btf.b\ t\ f)\,\mathbf{T}\,\mathbf{F}\,\mathbf{F} \\ &\twoheadrightarrow_\beta \mathbf{T}\,\mathbf{F}\,\mathbf{F} = (\lambda xy.x)\,\mathbf{F}\,\mathbf{F} \twoheadrightarrow_\beta \mathbf{F} \end{aligned}$$

$$\begin{aligned} \mathbf{or}\,\mathbf{F}\mathbf{T} &= (\lambda b_1 b_2.\mathbf{if}\ b_1\ \mathbf{T}\ b_2)\,\mathbf{F}\mathbf{T} \twoheadrightarrow_\beta \mathbf{if}\,\mathbf{F}\,\mathbf{T}\,\mathbf{T} = (\lambda btf.b\ t\ f)\,\mathbf{F}\,\mathbf{T}\,\mathbf{T} \\ &\twoheadrightarrow_\beta \mathbf{F}\,\mathbf{T}\,\mathbf{T} = (\lambda xy.y)\,\mathbf{T}\,\mathbf{T} \twoheadrightarrow_\beta \mathbf{T} \end{aligned}$$

$$\begin{aligned} \mathbf{not}\,\mathbf{T} &= (\lambda b_1.\mathbf{if}\ b_1\ \mathbf{F}\,\mathbf{T})\,\mathbf{T} \twoheadrightarrow_\beta \mathbf{if}\,\mathbf{T}\,\mathbf{F}\,\mathbf{T} = (\lambda btf.b\ t\ f)\,\mathbf{T}\,\mathbf{F}\,\mathbf{T} \\ &\twoheadrightarrow_\beta \mathbf{T}\,\mathbf{F}\,\mathbf{T} = (\lambda xy.x)\,\mathbf{F}\,\mathbf{T} \twoheadrightarrow_\beta \mathbf{F} \end{aligned}$$

Numere naturale

Numere naturale

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

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

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

$$
\begin{array}{rcl c l}
\overline{0} & \triangleq & \lambda fx.f^0\,x & = & \lambda fx.x \\
\overline{1} & \triangleq & \lambda fx.f^1\,x & = & \lambda fx.f\,x \\
\overline{2} & \triangleq & \lambda fx.f^2\,x & = & \lambda fx.f\,(f\,x) \\
\overline{3} & \triangleq & \lambda fx.f^3\,x & = & \lambda fx.f\,(f\,(f\,x)) \\
 & \vdots & & & \\
\overline{n} & \triangleq & \lambda fx.f^n\,x & = & \lambda fx.\underbrace{f(f(\ldots(f}_{n}\;x)\ldots))
\end{array}
$$

Numere naturale

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

Acum putem defini funcția succesor prin

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

Observați că **Succ** pe argumentul $\overline{n}$ returnează o funcție care primește ca argument o funcție f, îi aplică $\overline{n}$ 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.

$$\begin{array}{rcl}
\textbf{Succ}\,\overline{n} & = & (\lambda nfx.f\,(n\,f\,x))\,\overline{n} \\
& \twoheadrightarrow_\beta & \lambda fx.f\,(\overline{n}\,f\,x) \\
& \twoheadrightarrow_\beta & \lambda fx.f\,(f^n\,x) \\
& = & \lambda fx.f^{n+1}\,x \\
& = & \overline{n+1}
\end{array}$$

Numere naturale

$$\overline{n} \triangleq \lambda fx.f^n\, x \qquad\qquad \textbf{Succ} \triangleq \lambda nfx.f\,(n\,f\,x)$$

Putem face operații aritmetice de bază cu numeralii Church.

Pentru adunare, putem defini

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

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

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

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

Numere naturale

$$\overline{n} \triangleq \lambda fx.f^n\, x \qquad \textbf{Succ} \triangleq \lambda nfx.f\,(n\, f\, x)$$

Putem defini adunarea și ca aplicarea repetată a funcției succesor:

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

$$\begin{aligned}
\textbf{add}'\,\overline{m}\,\overline{n} &= (\lambda mn.m\,\textbf{Succ}\,n)\,\overline{m}\,\overline{n} \\
&\twoheadrightarrow_\beta \overline{m}\,\textbf{Succ}\,\overline{n} \\
&= (\lambda fx.f^m\,x)\,\textbf{Succ}\,\overline{n} \\
&\twoheadrightarrow_\beta \textbf{Succ}^m\,\overline{n} \\
&= \underbrace{\textbf{Succ}(\textbf{Succ}(\ldots(\textbf{Succ}}_{m}\,\overline{n})\ldots)) \\
&\twoheadrightarrow_\beta \underbrace{\textbf{Succ}(\textbf{Succ}(\ldots(\textbf{Succ}}_{m-1}\,\overline{n+1})\ldots)) \\
&\twoheadrightarrow_\beta \overline{m+n}
\end{aligned}$$

Numere naturale

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

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

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

$$\begin{aligned} \mathbf{mul} &\triangleq \lambda mn.m\,(\mathbf{add}\,n)\,\overline{0} \\ \mathbf{exp} &\triangleq \lambda mn.m\,(\mathbf{mul}\,n)\,\overline{1} \end{aligned}$$

Numere naturale

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

$$\begin{aligned}\textbf{isZero}(0) &= \textit{true} \\ \textbf{isZero}(n) &= \textit{false} \qquad \text{dacă } n \neq 0\end{aligned}$$

O codare în lambda calcul a unei astfel de funcții este

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

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 λ-termeni.

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

De exemplu, să considerăm funcția factorial

$$
\begin{aligned}
0! &= 1 \\
n! &= n \cdot (n-1)!, \qquad \text{dacă } n \neq 0
\end{aligned}
$$

Puncte fixe

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 $f(x) = x$.

β-echivalență

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

$\twoheadrightarrow_\beta$ 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 β-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 λ-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 λ-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

$$\begin{array}{rcl} M & = & (\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x)) \\ & \to_\beta & F\,((\lambda x.F\,(x\,x))\,(\lambda x.F\,(x\,x))) \\ & = & F\,M \end{array}$$

Deci avem $FM =_\beta M$.

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 $\mathbf{Y}F \twoheadrightarrow_\beta F\,(\mathbf{Y}\,F)$.
- Combinatorul de punct fix al lui Turing
  $\Theta \triangleq (\lambda xy.y\,(x\,x\,y))\,(\lambda xy.y\,(x\,x\,y))$
  Pentru orice termen F, ΘF este un punct fix al lui F deoarece $\Theta F \twoheadrightarrow_\beta F\,(\Theta\,F)$.

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

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

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

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

Notăm termenul $\lambda fn.\mathbf{if}\,(\mathbf{isZero}\,n)\,(\overline{1})\,(\mathbf{mul}\,n\,(f(\mathbf{pred}\,n))\,)$ 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 $\mathbf{Y}\,F \twoheadrightarrow_\beta F\,(\mathbf{Y}\,F)$), de aceea putem rezolva ecuația de mai sus luând

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

Observați că **fact** a dispărut din partea dreaptă.

Exercițiu. Evaluați $\mathbf{fact}\,\overline{2}$ ținând cont că $\mathbf{fact} \twoheadrightarrow_\beta F\,\mathbf{fact}$.

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:

- Aplicări de forma *x x* sau *M M* sunt pemise, deși sunt contraintuitive.
- Existența formelor normale pentru λ-termeni nu este garantată și putem avea "calcule infinite" nedorite
- Orice λ-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 $\mathbb{V} = \{\alpha, \beta, \gamma, \ldots\}$ o mulțime infinită de tipuri variabilă.

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

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

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

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

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

- $Nat \to Real$, mulțimea tuturor funcțiilor de la numere naturale la numere reale
- $(Nat \to Int) \to (Int \to Nat)$, 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 $\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T}$

Exemple de tipuri simple:

- γ
- $(\beta \to \gamma)$
- $((\gamma \to \alpha) \to (\alpha \to (\beta \to \gamma)))$

În tipurile săgeată, parantezele exterioare pot fi omise.

Parantezele în tipurile săgeată sunt asociative la dreapta.

De exemplu,

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

Termeni și tipuri

Ce înseamnă că un termen M are un tip σ?

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

Termeni și tipuri

Ce înseamnă că un termen M are un tip σ?

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

Variabilă. Dacă o variabilă x are un tip σ, 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 σ?

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

Variabilă. Dacă o variabilă x are un tip σ, 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 $M : \sigma \to \tau$, iar N trebuie să fie "adecvat" pentru această funcție, adică $N : \sigma$.

$$\text{Dacă } M : \sigma \to \tau \text{ și } N : \sigma, \text{ atunci } M\,N : \tau.$$

Termeni și tipuri

Ce înseamnă că un termen M are un tip σ?

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

Variabilă. Dacă o variabilă x are un tip σ, 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 $M : \sigma \to \tau$, iar N trebuie să fie "adecvat" pentru această funcție, adică $N : \sigma$.

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

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

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

Termeni și tipuri

Variabilă. $x : \sigma$.

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

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

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

Termeni și tipuri

Variabilă. $x : \sigma$.

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

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

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

Exemple.

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

Termeni și tipuri

Variabilă. $x : \sigma$.

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

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

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

Exemple.

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

Termeni și tipuri

Variabilă. $x : \sigma$.

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

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

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

Exemple.

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

Discuție despre asocitativitate

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

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

Discuție despre asocitativitate

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

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

$$\begin{aligned} &f : \rho \to \sigma \to \tau \\ &f\,x\,y : \tau \end{aligned}$$

  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 prescriere 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. $x : \alpha \to \alpha$
2. $y : (\alpha \to \alpha) \to \beta$
3. $z : \beta$
4. $u : \gamma$

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

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

Church-typing vs. Curry-typing

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

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

1. $x : \alpha \to \alpha$
2. $y : (\alpha \to \alpha) \to \beta$
3. $z : \beta$
4. $u : \gamma$

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

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

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

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

Din (3) și (6), prin abstractizare obținem (7): $\lambda zu.\, z : \beta \to \gamma \to \beta$.
Nu uitați că $\beta \to \gamma \to \beta$ înseamnă $\beta \to (\gamma \to \beta)$.

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

Church-typing vs. Curry-typing

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

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

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

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

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

Church-typing vs. Curry-typing

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

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

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

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

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

- Observăm că M este o aplicare a lui $\lambda zu.\, z$ termenului $y\, x$.
- Atunci $\lambda zu.\, z$ trebuie să aibă un tip săgeată, de exemplu $\lambda zu.\, z : A \to B$, 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 $M = (\lambda zu.\, z)\,(y\, x)$ și am dedus până acum:

$$\lambda zu.\, z : A \to B \qquad y\, x : A \qquad M : B$$

- Faptul că $\lambda zu.\, z : A \to B$ implică că $z : A$ și $\lambda u.\, z : B$.
- Deducem că B este tipul unei abstractizări, deci $B \equiv C \to D$, 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 $y : E \to F$ 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:

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

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

$$(*) \qquad x : E \qquad y : E \to A \qquad z : A \qquad u : C$$

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

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

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

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

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

Church-typing vs. Curry-typing

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

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

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

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

- $x : \beta, \quad y : \beta \to \alpha, \quad z : \alpha, \quad u : \delta, \quad M : \delta \to \alpha$
- $x : \alpha \to \alpha, \quad y : (\alpha \to \alpha) \to \beta, \quad z : \beta, \quad u : \gamma, \quad M : \gamma \to \beta$
  (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

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

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

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

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.

Church-typing

Exemplu. Să considerăm exemplul anterior $(\lambda zu.\, z)\,(y\, x)$.

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 $(\lambda zu.\, z)\,(y\, x)$.

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 $x : \alpha \to \alpha$ și $y : (\alpha \to \alpha) \to \beta$, atunci folosim notația:

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

Church-typing

Exemplu. Să considerăm exemplul anterior $(\lambda zu.\, z)\,(y\,x)$.

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 $x : \alpha \to \alpha$ și $y : (\alpha \to \alpha) \to \beta$, atunci folosim notația:

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

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

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

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

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

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

Mulțimea λ-termenilor cu pre-tipuri $\Lambda_{\mathbb{T}}$ este

$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\, \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\, \Lambda_{\mathbb{T}}$$

O afirmație este o expresie de forma $M : \sigma$, unde $M \in \Lambda_{\mathbb{T}}$ și $\sigma \in \mathbb{T}$. Într-o astfel de afirmație, M se numește subiect și σ 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 judecată este o expresie de forma $\Gamma \vdash M : \sigma$, unde Γ 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 σ în contextul Γ.

Sistem de deducție pentru calculul Church $\lambda\!\to$

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

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau} \; (app)$$

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

Un termen M în calculul $\lambda\!\to$ este legal dacă există un context Γ și un tip ρ astfel încât $\Gamma \vdash M : \rho$.

Sistem de deducție pentru calculul Church $\lambda{\rightarrow}$

$$\frac{}{\Gamma \vdash x : \sigma}\ (var) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}\ (app) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma.\, M) : \sigma \to \tau}\ (abs)$$

dacă $x : \sigma \in \Gamma$

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

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

Sistem de deducție pentru calculul Church $\lambda{\to}$

$$\frac{}{\Gamma \vdash x : \sigma}\ (var) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}\ (app) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma.\, M) : \sigma \to \tau}\ (abs)$$

dacă $x : \sigma \in \Gamma$

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta}\ (var) \qquad \dfrac{}{y : \alpha \to \beta, z : \alpha \vdash z : \alpha}\ (var)}{y : \alpha \to \beta, z : \alpha \vdash (yz) : \beta}\ (app)}{y : \alpha \to \beta \vdash (\lambda z : \alpha.\, yz) : \alpha \to \beta}\ (abs)}{\emptyset \vdash (\lambda y : \alpha \to \beta.\, \lambda z : \alpha.\, yz) : (\alpha \to \beta) \to \alpha \to \beta}\ (abs)$$

Diferite stiluri pentru a scrie deducții

În exemplul anterior, am scris derivarea în stilul arbore.

În stilul liniar, derivarea precedentă ar arăta astfel:

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

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

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.

$$\boxed{\begin{array}{ll} y : \alpha \to \beta & (context) \\ (\lambda z : \alpha .\, yz) : \alpha \to \beta & \end{array}}$$

$$(\lambda y : \alpha \to \beta .\, \lambda z : \alpha .\, yz) : (\alpha \to \beta) \to \alpha \to \beta \quad (abs)$$

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.

$$
\begin{array}{|l}
y : \alpha \to \beta \qquad\qquad\qquad (\mathit{context}) \\
\begin{array}{|l}
z : \alpha \qquad (\mathit{context}) \\
(yz) : \beta
\end{array} \\
(\lambda z : \alpha .\, yz) : \alpha \to \beta \qquad (\mathit{abs})
\end{array}
$$

$$(\lambda y : \alpha \to \beta .\, \lambda z : \alpha .\, yz) : (\alpha \to \beta) \to \alpha \to \beta \quad (\mathit{abs})$$

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.	$y : \alpha \to \beta$	$(\mathit{context})$
2.	$z : \alpha$	$(\mathit{context})$
3.	$(yz) : \beta$	(app) cu 1 și 2
4.	$(\lambda z : \alpha.\, yz) : \alpha \to \beta$	(abs) cu 3
5.	$(\lambda y : \alpha \to \beta.\, \lambda z : \alpha.\, yz) : (\alpha \to \beta) \to \alpha \to \beta$	(abs) cu 4

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

Exercițiu. Arătăți că termenul $\lambda x : ((\alpha \to \beta) \to \alpha).\, x\,(\lambda z : \alpha.\, y)$ are tipul $((\alpha \to \beta) \to \alpha) \to \alpha$ 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 $\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T}$

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

Mulțimea λ-termenilor cu pre-tipuri $\Lambda_{\mathbb{T}}$ $\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\,\Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\,\Lambda_{\mathbb{T}}$

- O afirmație este o expresie de forma $M : \sigma$, unde $M \in \Lambda_{\mathbb{T}}$ și $\sigma \in \mathbb{T}$.
- O declarație este o afirmație de forma $x : \sigma$.
- Un context Γ 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 $\lambda\!\to$

$$\frac{}{\Gamma \vdash x : \sigma}\ (var) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}\ (\to_E) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma.\, M) : \sigma \to \tau}\ (\to_I)$$

dacă $x : \sigma \in \Gamma$

Un termen M în calculul $\lambda\!\to$ 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

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

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.

$$? \vdash term : ?$$

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

$$context \vdash term : ?$$

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.

$$\textit{context} \vdash \; ? : \textit{type}$$

Toate aceste probleme sunt decidabile pentru calculul Church $\lambda{\to}$!

Limitări ale lambda-calculului cu tipuri simple

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

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

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

De exemplu, $\mathbf{add} \triangleq \lambda mnfx.\,m\,f\,(n\,f\,x)$ 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 $(\lambda f.\, \mathbf{if}\,(f\,\mathbf{T})\,(f\,3)\,(f\,5))\,(\lambda x.\, x)$ 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
$$\begin{array}{l}\mathbf{let}\ f = \lambda x.\, x\ \mathbf{in}\\ \mathbf{if}\,(f\,\mathbf{T})\,(f\,3)\,(f\,5)\end{array}$$
- Cuantificatori de tipuri. De exemplu, am avea
$$\lambda x.\, x : \Pi\alpha\, .\, \alpha \to \alpha$$
  Operatorul de legare Π face explicit faptul că variabila de tip α nu este rigidă.

Alte tipuri

Tipul Unit și constructorul unit

Mulțimea tipurilor

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \mathtt{Unit}$$

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

$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\, \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\, \Lambda_{\mathbb{T}} \mid \mathrm{unit}$$

$$\frac{}{\Gamma \vdash \mathrm{unit} : \mathtt{Unit}}\ (\mathit{unit})$$

Tipul Void

Mulțimea tipurilor

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \texttt{Unit} \mid \texttt{Void}$$

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

$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\, \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\, \Lambda_{\mathbb{T}} \mid \mathrm{unit}$$

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

Tipul produs și constructorul pereche

Mulțimea tipurilor

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \texttt{Unit} \mid \texttt{Void} \mid \mathbb{T} \times \mathbb{T}$$

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

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

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

Tipul produs și constructorul pereche

Mulțimea tipurilor

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \texttt{Unit} \mid \texttt{Void} \mid \mathbb{T} \times \mathbb{T}$$

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

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

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

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

Tipul sumă și constructorii Left/Right

Mulțimea tipurilor

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \texttt{Unit} \mid \texttt{Void} \mid \mathbb{T} \times \mathbb{T} \mid \mathbb{T} + \mathbb{T}$$

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

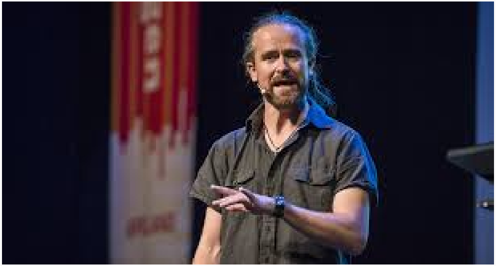
$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\, \Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\, \Lambda_{\mathbb{T}} \mid \text{unit} \mid \langle \Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}} \rangle \mid \mathit{fst}\ \Lambda_{\mathbb{T}} \mid \mathit{snd}\ \Lambda_{\mathbb{T}}$$
$$\mid \text{Left}\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

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \text{Left}\ M : \sigma + \tau}\ (+_{I_1}) \qquad \frac{\Gamma \vdash M : \tau}{\Gamma \vdash \text{Right}\ M : \sigma + \tau}\ (+_{I_2})$$

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

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

Un program simplu în Haskell

```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
makePoint x y = Point x y

getX :: Point -> Int
getX (Point x y) = x

getY :: Point -> Int
getY (Point x y) = y
```

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

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

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

Generalizare

$$\frac{x : Int \quad y : Int}{makePoint\ x\ y : Point}\ (Point_I) \qquad \frac{M : \sigma \quad N : \tau}{\langle M, N \rangle : \sigma \times \tau}\ (\times_I)$$

$$\frac{p : Point}{getX\ p : Int}\ (Point_{E_1}) \qquad \frac{M : \sigma \times \tau}{fst\ M : \sigma}\ (\times_{E_1})$$

$$\frac{p : Point}{getY\ p : Int}\ (Point_{E_2}) \qquad \frac{M : \sigma \times \tau}{snd\ M : \tau}\ (\times_{E_2})$$

Generalizare

$$\frac{x : Int \quad y : Int}{makePoint\ x\ y : Point}\ (Point_I) \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau}\ (\times_I)$$

$$\frac{M : Point}{getX\ M : Int}\ (Point_{E_1}) \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash fst\ M : \sigma}\ (\times_{E_1})$$

$$\frac{M : Point}{getY\ M : Int}\ (Point_{E_2}) \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash snd\ M : \tau}\ (\times_{E_2})$$

Alt exemplu simplu

```
f = (\x -> x * 3) :: Int -> Int
```

$$\frac{\{x : Int\} \vdash x * 3 : Int}{\lambda x.x * 3 : Int \to Int} \ (fun_I)$$

```
> f 5
15
```

$$\frac{f : Int \to Int \quad 5 : Int}{f\ 5 : Int} \ (fun_E)$$

Generalizare

$$\frac{\{x : Int\} \vdash x * 3 : Int}{\lambda x.x * 3 : Int \to Int}\ (fun_I) \qquad \frac{\{x : \sigma\} \vdash M : \tau}{\lambda x.M : \sigma \to \tau}\ (\to_I)$$

$$\frac{f : Int \to Int \quad 5 : Int}{f\ 5 : Int}\ (fun_E) \qquad \frac{M : \sigma \to \tau \quad N : \sigma}{M\ N : \tau}\ (\to_E)$$

Generalizare

$$\frac{\{x : Int\} \vdash x * 3 : Int}{\lambda x.x * 3 : Int \to Int}\ (fun_I)$$

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

$$\frac{f : Int \to Int \quad 5 : Int}{f\ 5 : Int}\ (fun_E)$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}\ (\to_E)$$

Logica. Ce este adevăt și ce este fals?

Hai să schimbăm perspectiva iar!

Logica. Ce este adevăt și ce este fals?

Hai să schimbăm perspectiva iar!

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

σ = afară este întuneric
τ = porcii zboară

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

Logica. Ce este adevăt și ce este fals?

Hai să schimbăm perspectiva iar!

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

σ = afară este întuneric
τ = porcii zboară

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

Este adevărată această afirmație? Da!

σ	τ	$\tau \supset \sigma$	$\sigma \supset (\tau \supset \sigma)$
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 $e : V \to \{0, 1\}$.

Putem să o extindem o evaluare la formule:

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

σ	τ	$\sigma \wedge \tau$
0	0	0
0	1	0
1	0	0
1	1	1

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

σ	τ	$\sigma \supset \tau$
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.

Sintaxa unei logici

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).

$$\frac{\Gamma \vdash \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma \wedge \tau}\ (\wedge_I) \qquad \frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \sigma}\ (\wedge_{E_1}) \qquad \frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \tau}\ (\wedge_{E_2})$$

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

Arată cunoscut?

Corespondența Curry-Howard

λ-calcul cu tipuri	Deducție naturală
$\dfrac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau}\ (\times_I)$	$\dfrac{\Gamma \vdash \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \sigma \wedge \tau}\ (\wedge_I)$
$\dfrac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathit{fst}\ M : \sigma}\ (\times_{E_1})$	$\dfrac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \sigma}\ (\wedge_{E_1})$
$\dfrac{\Gamma \vdash p : \sigma \times \tau}{\Gamma \vdash \mathit{snd}\ p : \tau}\ (\times_{E_2})$	$\dfrac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \tau}\ (\wedge_{E_2})$
$\dfrac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to_I)$	$\dfrac{\Gamma \cup \{\sigma\} \vdash \tau}{\Gamma \vdash \sigma \supset \tau}\ (\supset_I)$
$\dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}\ (\to_E)$	$\dfrac{\Gamma \vdash \sigma \supset \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau}\ (\supset_E)$

Propositions are types! ♡

Să analizăm mai atent

λ-calcul cu tipuri	Deducție naturală
$\Gamma \vdash M : \sigma$	$\Gamma \vdash \sigma$

Faptul că există un termen de tip σ *(inhabitation of type σ)* înseamnă că σ este teoremă/are o demonstrație în logică! ♡

Să analizăm mai atent

λ-calcul cu tipuri

$$\frac{\{x : \sigma\} \vdash x : \sigma}{\vdash \lambda x.\, x : \sigma \to \sigma}\ (\to_I)$$

Deducție naturală

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

Să analizăm mai atent

λ-calcul cu tipuri

$$\frac{\{x:\sigma\} \vdash x:\sigma}{\vdash \lambda x.\, x:\sigma \to \sigma}\ (\to_I)$$

$$\cfrac{\cfrac{\overline{\{x:\sigma, y:\tau\} \vdash x:\sigma}}{\{x:\sigma\} \vdash \lambda y.\, x:\tau \to \sigma}\ (\to_I)}{\vdash \lambda x.\, (\lambda y.\, x):\sigma \to (\tau \to \sigma)}\ (\to_I)$$

Deducție naturală

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

$$\cfrac{\cfrac{\overline{\{\sigma, \tau\} \vdash \sigma}}{\{\sigma\} \vdash \tau \to \sigma}\ (\supset_I)}{\vdash \sigma \to (\tau \to \sigma)}\ (\supset_I)$$

Să analizăm mai atent

λ-calcul cu tipuri

$$\frac{\{x:\sigma\} \vdash x:\sigma}{\vdash \lambda x.\, x:\sigma \to \sigma}\ (\to_I)$$

$$\frac{\dfrac{\overline{\{x:\sigma, y:\tau\} \vdash x:\sigma}}{\{x:\sigma\} \vdash \lambda y.\, x:\tau \to \sigma}\ (\to_I)}{\vdash \lambda x.\,(\lambda y.\, x):\sigma \to (\tau \to \sigma)}\ (\to_I)$$

Deducție naturală

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

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

Proofs are Terms! ♡

Demonstrațiile sunt termeni!

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului σ	demonstrație a lui σ

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului σ	demonstrație a lui σ
tip produs	conjuncție
tip funcție	implicație

Corespondența Curry-Howard

Teoria Tipurilor	Logică
tipuri	formule
termeni	demonstrații
inhabitation a tipului σ	demonstrație a lui σ
tip produs	conjuncție
tip funcție	implicație
tip sumă	disjuncție
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)

Corespondența Curry-Howard

Inițial, corespondența Curry-Howard a fost între

Calculul Church $\lambda \rightarrow$	Sistemul de deducție naturală al lui Gentzen pentru logica intuiționistă

De ce?

- 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!