Prerequisites

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

2.1

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

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

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

2.2

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

$x[N/x]$	$\equiv$	N	
$y[N/x]$	$\equiv$	y	dacă $x \neq y$
$(M\,P)[N/x]$	$\equiv$	$(M[N/x])\,(P[N/x])$	
$(\lambda x.M)[N/x]$	$\equiv$	$\lambda x.M$	
$(\lambda y.M)[N/x]$	$\equiv$	$\lambda y.(M[N/x])$	dacă $x \neq y$ și $y \notin FV(N)$
$(\lambda y.M)[N/x]$	$\equiv$	$\lambda y'.(M\langle y'/y\rangle[N/x])$	dacă $x \neq y$, $y \in FV(N)$ și y' variabilă nouă

3.1

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

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

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

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

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

3.2

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).

5.1

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

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

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

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

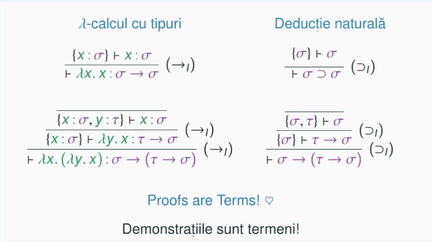
6.1

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! ♡

6.2



Quiz-uri

1.1. Care ar fi cea mai apropiata scriere in lambda calcul pentru A, unde $f(x) = x^2 + 1$ si $A = f(2)$?

5
$(x^2 + 1)(2)$
$2^2 + 1$
$(\lambda x.\ x^2 + 1)(2)$

1.2. Care din conceptele de mai jos nu este un model de calculabilitate?

masinile Turing
punctele fixe
funcțiile recursive
lambda calcul

1.3. In lambda calcul fara tipuri

trebuie sa specificam mereu tipul oricarei expresii
sunt eliminate expresiile de forma f(f)
nu specificam domeniul/codomeniul functiilor
putem avea efecte laterale

2.1. Care din lambda termenii de mai jos nu este închis?

λxyz.xxy
λxy.xxy
λx.xxy
λx.xx

2.2. Care sunt variabilele libere din termenul λx.xxy?

termenul nu are variabile libere
x
y
x și y

Quiz-uri (cont)

2.3. Care din următoarele afirmații este adevărată?

un combinator este orice lambda termen
un combinator este un lambda termen fără variabile libere (închis)
un combinator este un lambda termen cu variabile libere
un combinator este un lambda termen care are și variabile libere, și variabile legate

3.1. Ce este un β-redex?

un λ-termen de forma M[N/x]
un λ-termen de forma (λx.M)N
un λ-termen de forma (λx.M)
un λ-termen de forma x

3.2. Ce este o formă normală pentru un λ-termen?

un λ-redex
cea mai mică β-reducție
o α-echivalență
un λ-termen fără redex-uri

3.3. În ce constă strategia normală de evaluare pentru λ-termeni?

alegerea redex-ului cel mai din stânga și apoi cel mai din exterior
alegerea redex-ului cel mai din stânga și apoi cel mai din interior
aplicarea unei reduceri în corpul unei abstractizări
nu este definită

4.1. O codare în lambda calcul pentru constanta booleana true este:

λxy.x
λxy.xy
λx.x
nu se poate coda în lambda calcul

4.2. Codarea numerelor naturale în lambda calcul se mai numește și:

mașina Turing universală
numerele naturale nu se pot coda în lambda calcul
numeralii Church
redex

4.3. Un lambda termen M este punct fix al unui lambda termen F dacă

$F =_\beta M$
$F\ M =_\beta M$
$M\ F =_\beta M$
$F\ M =_\beta F$

Quiz-uri (cont)

5.1. Ce înseamnă că un termen M este typable?

există un tip σ astfel încât M să aibă tipul σ
există o derivare a lui M
M are o formă normală
M este o abstractizare

5.2. Care din următorii termeni este typable?

xx
xxy
x(xy)
niciunul din termenii de mai sus

5.3. Ce este o judecată în calculul λ→?

o expresie de forma M:σ
o expresie de forma x:σ
o expresie de forma Γ⊢M:σ
o abstractizare

6.1. Ce înseamnă type checking?

pentru un termen dat, constă în găsirea unui tip pentru un termen
pentru un termen dat, constă în găsirea unui context pentru un termen
pentru un context, termen și tip date, constă în verificarea faptului că termenul poate avea tipul în contextul dat
pentru un context și un tip date, constă în găsirea unui tip pentru termen în contextul dat

6.2.Care din problemele de mai jos nu este decidabilă pentru lambda calcul cu tipuri simple?

inhabitation
typability
type checking
toate de mai sus sunt probleme decidabile

6.3. Care din afirmațiile de mai jos este adevărată pentru tipul Void?

are un inhabitant numit void
nu poate exista un astfel de tip
nu are niciun inhabitant
orice termen poate avea tipul Void

Curs 4

Expresivitatea λ-calculului

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

Booleeni

•T ≜ λxy.x (dintre cele doua alternative o alege pe prima)
•F ≜ λxy.y (dintre cele doua alternative o alege pe a doua)
if = λbtf.t , if b = true
λbtf.f, if b = false
altfel spus, **if ≜ λbtf.b t f**
and ≜ λb1b2.if b1 b2 (sau λb1b2.b2 b1 b2 sau λb1b2.b1 b2 F)
or ≜ λb1b2.if b1 T b2
not ≜ λb1.if b1 F T F
Operațiile lucrează corect doar dacă primesc ca argumente valori booleene. Folosind lambda calcul fară tipuri, avem garbage in, garbage out.

Numere naturale

Numeralul Church pentru numărul n ∊ N este notat n(bar).
Numeralul Church n(bar) este λ-termenul λfx.f^nx, unde f^n reprezintă compunerea lui f cu ea însăși de n ori. Avem deci **n ≜ λfx.f^nx**.
Definim **Succ ≜ λnfx.f (n f x)**
add ≜ λmnfx.m f (n f x)
add' ≜ λmn.m *Succ* n
mul ≜ λmn.m (add n) 0(bar)
exp ≜ λmn.m (mul n) 1(bar)
isZero ≜ λnxy.n (λz.y) x

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 =β M.
Thm: În lambda calculul fără tipuri, orice termen are un punct fix.

Combinatori de punct fix

Curs 4 (cont)

= termeni închiși care „construiesc” un punct fix pentru un termen arbitrar.

Exemple:

Combinatorul de punct fix al lui Curry: **Y ≜ λy.(λx.y (x x)) (λx.y (x x))** (YF punct fix al lui F, pt orice teremn F, deoarece *YF →β F (Y F)*)

Combinatorul de punct fix al lui Turing: **Θ ≜ (λxy.y (x x y)) (λxy.y (x x y))** (ΘF punct fix al lui F, pt orice termen F, deoarece *ΘF →β F (Θ F)*)

Rezolvarea de ecuații în lambda calcul

fact n = if(isZero n) (1) (mul n (fact(pred n)))

fact = λn.if(isZero n) (1) (mul n (fact(pred n)))

fact = (λfn.if(isZero n) (1) (mul n (f(pred n))))fact

Notăm F = *λfn.if(isZero n) (1) (mul n (f(pred n)))*

Ultima ecuat, ie devine **fact = F fact**, o ecuație de punct fix.

Luăm deci **fact ≜ Y F**, adică **fact ≜ Y (λfn.if(isZero n) (1) (mul n (f(pred n))))**

Curs 1

Ce este o funcție în matematică?

extensional: „funcții prin grafice”, clasă mai largă de funcții, cuprinde și funcții care nu pot fi definite prin formule

(domeniu și codomeniu fixate, funcția e o mulțime de perechi -> duce intrări în ieșiri)

funcții extensional egale = pentru aceeași intrare obțin aceeași ieșire (*f(x) = g(x), ∀x∈ X*)

intensional: „funcții ca formule” (nu e mereu necesar să știm domeniul și codomeniul; paradigmă utilă în informatică->un program = o fct de la intrări la ieșiri)

funcții intensional egale: definite prin aceași formulă (eventual prelucrată)

Curs 1 (cont)

Lambda calcul

lambda calcul = teorie a *funcțiilor ca formule*

Exemplu: $\lambda x.x^2$ = $x\text{->}x^2$ (e expresie, nu instrucțiune/declarație)

Variabila **x** este *locală/legată* în termenul $\lambda x.x^2$

Exemplu: *f∘f* = λx.f(f(x))

f→f∘f = λf.λx.f(f(x))

$((\lambda f.\lambda x.f(f(x)))(\lambda y.y^2))(5) = 625$

Funcția identitate f = λx.x are tipul X->X, unde X poate să fie orice mulțime

Funcția g = λf.λx.f(f(x)) are tipul (X->X)->(X->X)

f(f) ≃ (λx.x)(λx.x) ≃ λx.x ≃ f

Combinatorul ω = λx.xx care reprezintă funcția care aplică x lui x

ω(λy.y) ≃ (λx.xx)(λy.y) ≃ (λy.y)(λy.y) ≃ (λy.y)

ω =λx.x(x), ω(ω) = ?

ω(f) = f(f), for any f that belongs to its domain. So ω(ω) = ω(ω) and cannot be further evaluated

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 expersii

-- nu putem aplica funcții unui argument care are alt tip față de denumirea funcției

-- expresii de forma *f(f)* sunt eliminate, chiar dacă f e funția identitate

Lambda calcul cu tipuri polimorfice

Curs 1 (cont)

-- o situație intermediară între cele două de mai sus
-- putem specifica că o expresie are tipul X->X, dar fără a specifica cine este X

Calculabilitate

informal: o funcție este calculabilă dacă există o metodă „pen-and--paper” care permite calcularea lui *f(n)*, pentru orice n
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
Godel - a definit clasa *funcțiilor recursive* și a postualt că o funcție este calculabilă ddacă este o funcție recursivă
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 labda termen
Teza Church-Turing - cele 3 metode sunt echivalente și coincid cu noțiunea intuitivă de calculabilitate

Ce este o demonstrație?

Logica clasică: plecând de la niște presupuneri, este suficient să ajungi la o contradicție
Logica constructivă: pentru a arăta că un obiect există, trebuie să îl construim explicit
- legătura dintre lambda calcul și logica constructivă este dată de paradigma „proof-as-programs” (o demonstrație trebuie să fie o construcție, un program; lambda calculul este o notație pentru un astfel de program)

Curs 2

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*

Lambda termeni

Fie V o mulțime infinită de variabile, notate *x, y, z,....*
Mulțimea *lambda termenilor* este dată de următoarea formă BNF:
lambda termen = variabilă | aplicare | abstractizare
M, N ::= x | (M N) | (λx.M)
Definiție alternativă:
Fie V o mulțime infinită de variabile, notate *x, y, z, ...*
Fie A un alfabet format din elementele din V și simboluri speciale *(,), λ, ..*
Fie A *mulțimea tuturor cuvintelor finite pentru alfabetul A.*
Mulțimea ***lambda termenilor*** *este cea mai mică submulțime Λ ⊆ A* astfel încât:
[Variabila] V ⊆ Λ
[Aplicare] dacă M, N ∈ Λ atunci (M N) ∈ Λ
[Abstractizare] dacă x ∈ V și M ∈ Λ atunci (λx.M) ∈ Λ

Convenții

•Se elimină parantezele exterioare.
• Aplicarea este asociativa la stânga (*M N P = (M N P), f x y z = ((f x) y) z*)
• Corpul abstractizării (partea de după punct) se extinde la dreapta cât se poate (*λx.M N = λx.(M N)*, nu *(λx.M) N*)
• Mai mulți λ pot fi comprimați (*λxyz.M = λx.λy.λz.M*)
Exemple:
(λx.(λy.(λz.((x z)(y z))))) = λxyz.x z (y z)
(((a b) (c d)) ((e f) (g h))) = a b (c d) (e f (g h))
x x x x = (((x x) x) x)
λx.x λy.y = (λx.(x (λy.y)))

Curs 2 (cont)

Variabile libere și variabile legate

λx.N, unde

λ_._ = operator de legare (binder)

x (din λx._) = variabilă de legare (binding)

N = 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 (combinator).

Exemplu: M ≡ (λx.xy) (λy.yz)

x este legată

z este liberă,

y are 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 prin:

FV(x) = {x}

FV(M N) = FV(M) ∪ FV(N),

FV(λx.M) = FV(M)\{x}

Redenumire de variabile

Dacă x, y sunt variabile și M este un termen, M<y/x> este rezultatul obținut după redenumirea lui x cu y în M.

x<y/x>≡ y, z<y/x>≡ z, dacă x != z

(M N)<y/x> ≡ (M<y/x>) (N<y/x>),

(λx.M)<y/x> ≡ λy.(M<y/x>),

(λz.M)<y/x> ≡ λz.(M<y/x>), dacă x != z

α-echivalență

Curs 2 (cont)

α-echivalența = cea mai mică relație de congruență =α pe mulțimea lambda termenilor, aî pentru orice termen M și orice variabilă y care nu apare în M, avem λx.M =α λy.(M<y/x>)

-- cea mai mică relație pe lambda termeni care satisface regulile:

  vezi poza din cardul 2.1

Convenția Barendregt: variabilele legate sunt redenumite pt a fi distincte.

Substituții (card 2.2)

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

Cazuri speciale:

1. Înlocuim doar variabile libere: x (λxy.x)[N/x] = N (λxy.x), nu N (λxy.N) sau N (λNy.N).

2.Nu vrem sa legăm variabile libere neintenționat:M ≡ λx.y x, N ≡ λz.x z (x legată în M și lilberă în N). *Redenumim variabilele legate înainte de substituție!*

M[N/y] = (λx'.y x')[N/y] = λx'.N x'= λx'.(λz.x z) x'

Exemple

(λz.x)[y/x] = λz.y

(λy.x)[y/x] = λy'.y, nu λy.y,

(λy.x)[(λz.z w)/x] = λyz.zw

Curs 3

β-reducții (card 3.1)

Curs 3 (cont)

Convenție: M=N (2 termeni egali) dacă sunt α-echivalenți.
β-reducție = procesul de a evalua lambda termeni prin „pasarea de argumente funcțiilor"
β-redex = un termen de forma (λx.M) N
redusul unui redex (λx.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 până nu mai sunt redexuri
forma normală = un lambda termen fără redexuri
Observații:
•reducerea unui redex poate crea noi redex-uri/șterge alte redex-uri
• numărul de pași necesari pâna a atinge o fn poate varia, în funcție de ordinea în care sunt reduse redex-urile, iar rezultatul final nu depinde de alegerea redex-urilor
•lungimea unui termen nu trebuie să scadă în procesul de β-reducție; poate crește sau rămâne neschimbat.

β-forma normală

Curs 3 (cont)

(λx.x x) (λx.x x) nu poate fi redus la o β-formă normală (evaluarea nu se termină)
Există lambda termeni care deși pot fi reduși la o formă norală, pot să nu o atingă niciodată (contează *strategia de evaluare*).
Notăm cu M -»β M' faptul că M poate fi β-redus până în M' în 0 sau mai mulți pași (*închiderea reflexivă și tranzitivă a relației →β*)
Notăm cu *M =β 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.
=β este închiderea reflexivă, simetrică și tranzitivă a relației →β.
M este **slab normalizabil** (weakly normalising) dacă există N în forma normală aî M -»β N.
M este **puternic normalizabil** (strong normalising) dacă nu există reduceri infinite care încep din M.
puternic normalizabil => slab normalizabil
Exemple:
(λx.y) ((λz.zz) (λw.w)) - puternic normalizabil.
(λxy.y) ((λx.x x) (λx.x x)) (λz.z) - slab normalizabil, dar nu puternic normalizabil.

Confluența β-reducției. Teorema Church-Rosser

Card 3.2

*Exemple

(λx.x) M -»β M
(λxy.x) M N -»β M
(λx.x x) (λy.y y y) -»β (λy.y y y) (λy.y y y) (λy.y y y). . .

Strategii de evaluare

Curs 3 (cont)

Lambda calculul nu specifică o strategie de evaluare, fiind *nedeterminist*

Strategia normală = leftmost-outermost (alegem redex-ul cel mai din stânga și apoi cel mai din exterior)

dacă M1 și M2 sunt redex-uri și M1 este un subtermen al lui M2, atunci M1 nu va fi um redex ales.

printre redex-urile care nu sunt subtermeni ai altor redex-uri (și sunt incompatibili 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.

Strategia aplicativă = leftmost-innermost (alegem redex-ul cel mai din stânga și apoi cel mai din interior)

dacă M1 și M2 sunt redex-uri și M1 este un subtermen al lui M2, atunci M2 nu va fi următorul redex ales

printre redex-urile care nu sunt subtermeni ai altor redex-uri (și sunt incompatibili față de relația de subtermen), îl alegem pe cel mai din stânga.

Strategii în programare funcțională

Call-by-Name (CBN) = strategia normală fără a face reduceri în corpul unei λ-abstractizări.

Call-by-Value (CBV) = strategia aplicativă fără a face reduceri în corpul unei λ-abstractizări (folosit în general de limbajele de progr funcțională, excepție Haskell, e o formă de evaluare leneșă).

O *valoare* este un λ-term pt care nu există β-reducții date de strategia de evaluare considerată.

Curs 5

Tipuri simple

Fie V = {α, β, γ, ...} o mulțime infinită de *tipuri variabilă*.

Mulțimea tuturor *tipurilor simple* T este definită prin:

T = V | T → T

• (*Tipul variabilă*) Dacă α ∈ V, atunci α ∈ T.

• (*Tipul săgeată*) Dacă σ, τ ∈ T, atunci (σ → τ) ∈ T.

Tipurile săgeată reprezintă *tipuri pentru funcții* cum ar fi *Nat->Real* (fct de la nr nat la nr reale) sau *(Nat ->Int)->(Int->Nat)* (fcț care au ca intrare o fcț de la nr nat la nr întregi li produce o fcț de la întregi la nat)

Parantezele în tipurile săgeată sunt asociative la dreapta.

Termeni și tipuri

M are tip σ: *M :σ*

Variabilă x :σ. Pp că orice var din M are tip unic (Dacă x :σ și x : τ, atunci σ ≡ τ)

Aplicare: Dacă M :σ → τ și N:σ, atunci M N : τ.

Abstractizare Dacă x :σ și M : τ, atunci λx. M :σ → τ.

M *are tip* (este typeable) daca există un tip σ astfel încât M :σ.

Metode de a asocia tipuri variabilelelor

Asocierea explicită (Church-typing):

•constă în prescierea unui unic tip pt fiecare variabilă, la introducerea acesteia (tipuri stabilite explicit)

Asocierea implicită (Curry-typing):

• constă în a nu prescrie un tip pt fiecare variabilă, ci în a le lăsa „deschise” (teremenii typeable sunt descoperiți printr-un proces de căutare).

Sistem de deducție pentru Church λ→ (card 5.1)

Curs 5 (cont)

Mulțimea **λ-termenilor cu pre-tipuri ΛT** este: *ΛT = x / ΛT ΛT / λx : T. ΛT.*

*O **afirmație** este o expresie de forma M :σ, unde M ∊ ΛT și σ ∊ T (M =*subiect, *σ =*tip*).*

*O **declarație** este o afirmație în care subiectul e variabilă (x :σ*).*

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

O **judecată** este o expresie de forma Γ ⊢ M :σ, unde Γ este context și M :σ este o afirmație.

Curs 6

Ce probleme putem să rezolvăm în teoria tipurilor?

Type Checking

Se reduce la a verifica că putem găsi o derivare pentru

context ⊢ term : type

Well-typedness (Typability):

Se reduce la a verifica dacă un termen e legal.

? ⊢ term : ?

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

context ⊢ term : ?

Term Finding (Inhabitation)

Dându-se un context și un tip, să se stabilească dacă există un termen cu ac tip, în contextul dat

context ⊢ ? : type

!Toate aceste probleme sunt decidabile pentru calculul Church λ→!

Limitări ale lambda-calculului cu tipuri simple

•nu mai avem recursie nelimitată, dar avem recursie primitivă (looping cu număr cunoscut de iterații)

•tipurile pot fi prea restrictive (soluții posibile: *let-polymorphism*, unde variabilele libere se redenumesc la fiecare folosire; *cuantificatori de tipuri* operatorul de legare Π)

Corespondența Curry-Howard (card 6.1, 6.2)

Curs 6 (cont)

Teoria Tipurilor	Logică
◦ tipuri ◦ termeni ◦ inhabitation a tipului σ	◦formule ◦ demonstrații ◦ demonstrație a lui σ
◦ tip produs ◦ tip funcție ◦ tip sumă ◦ tipul void ◦ tipul unit	◦ conjuncție ◦ implicație ◦ disjuncție ◦ false ◦ true

Logica Intuiționistă

◦ logică constructivă, bazată pe *demonstrație*

◦ demonstrațiile sunt executabile și produc exemple

Urm formule echiv nu sunt demonstrabile în logica intuiționistă:

$\neg\neg\varphi \supset \varphi$ (dubla negație)

$\varphi \vee \neg\varphi$ (excluded middle)

$((\varphi \supset \tau) \supset \varphi) \supset \varphi$ (legea lui Pierce)

Nu există semantică cu tabele de adevăr pentru logica intuiționistă!

Inițial, corespondența Curry-Howard a fost între: (Calcul Church λ→) și (Sistemul de deducție naturală al lui Gentzen pentru logica intuiționistă)