FLP Cheatsheet

Lambda calcul

Permite manipularea funcțiilor ca expresii.

Funcția $f(x) = x^2$ devine $\lambda x.x^2$.

Spunem că variabila x este **legată** în termenul $\lambda x.x^2$.

Lambda calculul poate fi:

Fără tipuri

- nu specificăm tipul niciunei expresii
- nu specificăm domeniul / codomeniul funcțiilor
- flexibil, dar riscant

Cu tipuri simple

- specificăm tipul oricărei expresii
- argumentul unei funcții trebuie să facă parte din domeniul acesteia
- expresiile de forma f(f) sunt eliminate

Cu tipuri polimorfice

- o situație intermediară între celelalte două
- putem specifica că o expresie are tipul $X \to X$, fără a specifica cine e X

Calculabilitate

Următoarele definiții sunt echivalente (teza Church-Turing). O funcție e calculabilă ddacă:

- Poate fi calculată de o **mașină Turing**. (Turing)
- Este o **funcție recursivă**. (Gödel**)**
- Poate fi scrisă ca un **lambda termen** (Church).

Lambda termeni

- lambda termen = variabilă | aplicare | abstractizare
- $M, N ::= x \mid (MN) \mid (\lambda x.M)$

Convenții

- Aplicarea este asociativă la stânga: $MNP = (MN)P$, $f\ x\ y\ z = ((f\ x)\ y)\ z$
- Corpul abstractizării (partea de după punct) se extinde la dreapta cât se poate: $\lambda x.MN = (\lambda x.M)N$.
- Mai mulți λ pot fi comprimați: $\lambda x.\lambda y.\lambda z.M = \lambda xyz.M$

  Exemplu: $(\lambda x(\lambda y(\lambda z((x\ z)(y\ z))))) = \lambda xyz.x\ z\ (y\ z)$

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 al lui x; toate aparițiile lui x în N sunt legate

- O apariție care nu este legată se numește liberă
- Un termen fără variabile libere se numește închis (closed)
- Un termen închis se mai numește și combinator.

  Exemplu: Pentru $M \equiv (\lambda x.xy)(\lambda y.yz)$, x este legată, z este liberă, y are și o apariție legată, și una liberă, iar mulțimea variabilelor libere ale lui M este $\{y, z\}$.

Mulțimea variabilelor libere dintr-un termen M este notată $FV(M)$ și e definită prin:
$FV(x) = \{x\}$, $FV(MN) = FV(M) \bigcup FV(N)$, $FV(\lambda x.M) = FV(M) \setminus \{x\}$

Redenumire de variabile: $M\langle y/x \rangle$ reprezintă redenumirea lui x cu y în M.

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

$z\langle y/x \rangle \equiv z$, dacă $x \neq z$

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

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

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

α-echivalența este 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 = doi termeni sunt egali, modulo redenumire de variabile

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

Substituții:

Vrem să înlocuim variabile cu lambda termeni: $M[N/x]$ este rezultatul obținut după înlocuirea lui x cu N în M.

1. Vrem să înlocuim doar variabile libere (numele variabilelor nu trebuie să afecteze rezultatul substituției):

   De exemplu, $x(\lambda xy.x)[N/x]$ este echivalent cu $N(\lambda xy.x)$.

2. Nu vrem să legăm variabile libere neintenționat.

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

$x[N/x] \equiv N$; $y[N/x] \equiv y$ (dacă $x \neq y$);

$(M\ P)[N/x] = (M[N/x]\ P[N/x])$ $(\lambda x.M)[N/x] = \lambda x.M$;

$(\lambda y.M)[N/x] = \lambda y.(M[N/x])$ (dacă $x \neq y$ și $y \notin FV(N)$)

$(\lambda y.M)[N/x] = \lambda y'.(M\langle y'/y \rangle[N/x])$ (dacă $x \neq y$ și $y \notin FV(N)$)

De exemplu:

- $(\lambda z.x)[y/x]$ devine $\lambda z.y$
- $(\lambda y.x)[y/x]$ devine $\lambda y'.y$
- $(\lambda y.x)[(\lambda z.zw)/x]$ devine $\lambda yz.zw$

β-reducții

- Spunem că doi termeni sunt egali daca 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 $M[N/x]$
- formă normală = un lambda termen fără redex-uri

β-formă normală

Notăm cu $M \twoheadrightarrow_\beta M'$ faptul că M poate fi redus până la M' în 0 sau mai mulți pași.

- M este slab normalizabil dacă există N în forma normală a.î. $M \twoheadrightarrow_\beta N$
- M este puternic normalizabil dacă nu există reduceri infinte care încep din M.

De exemplu:

- $(\lambda x.y)((\lambda z.zz)(\lambda w.w))$ este puternic normalizabil.
- $(\lambda xy.y)((\lambda x.xx)(\lambda x.xx))(\lambda z.z)$ este slab normalizabil, dar nu puternic normalizabil.

Teorema Church-Rosser: Dacă $M \twoheadrightarrow_\beta M_1$ și $M \twoheadrightarrow_\beta M_2$ atunci există M' a.î. $M_1 \twoheadrightarrow_\beta M'$ și $M_2 \twoheadrightarrow_\beta M'$.

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

Exemple:

- $(\lambda x.x)M = M; \qquad (\lambda xy.x)MN = N$
- $(\lambda x.xx)(\lambda y.yyy) = (\lambda y.yyy)(\lambda y.yyy)(\lambda y.yyy)...$

Strategii de evaluare

Strategia de evualuare ne spune în ce ordine să facem pașii de reducție. Ordinea **contează** în evaluarea expresiilor. Lambda calculul nu specifică o strategie de evaluare, fiind nedeterminist. O strategie de evaluare este necesară în limbajele de programare pentru a rezolva nedeterminismul.

- **Strategia normală = leftmost-outermost**
  - alegem redexul cel mai din stânga și apoi cel mai din exterior
  - Dacă un termen are o formă normală, atunci strategia normală va converge la ea

    $\underline{(\lambda xy.y)((\lambda x.xx)(\lambda x.xx))}(\lambda z.z) \to_\beta \underline{(\lambda y.y)(\lambda x.x)} \to_\beta \lambda x.x$
- **Strategia aplicativă = leftmost-innermost**
  - alegem redex-ul cel mai din stânga și apoi cel mai din interior

    $(\lambda xy.y)\underline{((\lambda x.xx)(\lambda x.xx))}(\lambda z.z) \to_\beta (\lambda xy.y)((\lambda x.xx)(\lambda x.xx))(\lambda z.z) \to_\beta ...$
- **Strategia call-by-name (CBN)**
  - strategia normală fără a face reduceri în corpul unei λ-abstractizări
  - amânăm evaluarea argumentelor cât mai mult posibil, făcând reducții de la stânga la dreapta în expresie → strategia folosită de Haskell

  Exemplu: $(\lambda x.succ\ x)((\lambda y.succ\ y)\ 3) \twoheadrightarrow_\beta succ((\lambda y.succ\ y)\ 3) \twoheadrightarrow_\beta succ\ (succ\ 3))$

  $\to succ\ 4 \to 5$

- **Strategia call-by-value (CBV)**
  - strategia aplictativă fără a face reduceri în corpul unei λ-abstractizări
  - majoritatea limbajelor folosesc CBV, în afară de Haskell

  Exemplu: $(\lambda x.succ\ x)((\lambda y.succ\ y)\ 3) \twoheadrightarrow_\beta (\lambda x.succ\ x)\ (succ\ 3) \twoheadrightarrow_\beta (\lambda x.succ\ x)\ 4 \twoheadrightarrow_\beta succ\ 4 \to 5$

O **valoare** este un λ-termen 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.

Booleni

$T \triangleq \lambda xy.x \qquad F \triangleq \lambda xy.y$

Funcția $if \triangleq \lambda btf.b\ t\ f$ returnează t dacă $b = true$ și f dacă $b = false$.

- $\mathbf{and} \triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ b_2\ F$
- $\mathbf{or} \triangleq \lambda b_1 b_2.\mathbf{if}\ b_1\ T\ b_2$
- $\mathbf{not} \triangleq \lambda b_1.\mathbf{if}\ b_1\ F\ T$

Operațiile se comportă rezonabil dacă primul argument este boolean. Altfel, folosind lambda calcul fără tipuri, avem garbage in, garbage out.

Numere naturale

Numeralii Church: $\overline{n} = \lambda fx.f^n\ x$, unde f^n reprezintă compunerea lui f cu ea însăși de n ori.

$\overline{0} \triangleq \lambda fx.x;\quad \overline{1} \triangleq \lambda fx.f\ x;\quad \overline{2} \triangleq \lambda fx.f\ (f\ x);\quad \overline{3} \triangleq \lambda fx.f\ (f\ (f\ x))$ ....

Funcția succesor: $Succ \triangleq \lambda nfx.f\ (n\ f\ x) \qquad Succ\ \overline{n} = \overline{n+1}$

Operații aritmetice:

- $\mathbf{add} \triangleq \lambda mnfx.m\ f\ (n\ f\ x) \qquad \mathbf{add}\ \overline{m}\ \overline{n} = \overline{m+n}$
- $\mathbf{add'} \triangleq \lambda mn.m\mathbf{Succ}\ n$
- $\mathbf{mul} \triangleq \lambda mn.m\ (\mathbf{add}\ n)\ \overline{0}$
- $\mathbf{exp} \triangleq \lambda mn.m\ (\mathbf{mul}\ n)\ \overline{1}$
- $\mathbf{isZero} \triangleq \lambda nxy.n\ (\lambda z.y)\ x$

Puncte fixe

Am notat cu $M \twoheadrightarrow_\beta M'$ faptul că M poate fi redus până la M' în 0 sau mai mulți pași de β-reducție. $\twoheadrightarrow_\beta$ este închiderea reflexivă și tranzitivă a relației $\to_\beta$.

Notăm cu $M =_\beta M'$ faptul că M poate fi redus până la M' în 0 sau mai mulți pași de β-reducție, transformare în care pașii pot fi și întorși. $=_\beta$ este închiderea reflexivă, simetrică și tranzitivă a relației $\to_\beta$.

De exemplu, $(\lambda y.y\ v)\ z =_\beta (\lambda x.z\ x)\ v$, deoarece ambele pot fi β-reduse la $z\ v$.

Punct fix: Dacă F și M sunt λ-termeni, spunem că M este punct fix al lui F dacă $FM =_\beta M$.

Teoremă: În lambda calcul fără tipuri, orice termen are un punct fix.

Combinatorii de puncte fixe sunt termeni închiși care "construiesc" un pct fix pt. un termen arbitrar:

- Combinatorul de punct fix al lui Curry: $\mathbf{Y} = \lambda y.(\lambda x.y(x\ x))\ (\lambda x.y(x\ x))$
  - 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: $\mathbf{\Theta} = (\lambda xy.y(x\ x\ y))\ (\lambda xy.y(x\ x\ y))$

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

Lambda calcul cu tipuri simple

- $V = \{\alpha, \beta, \gamma, ...\}$ mulțime infinită de tipuri variabilă
- Mulțimea tipurilor simple este $T = V \mid T \to T$.
  - (Tipul variabilă) Dacă $\alpha \in V$, atunci $\alpha \in T$.
  - (Tipul săgeată) Dacă $\sigma, \tau \in T$, atunci $\sigma \to \tau \in T$.

  Exemplu de tip simplu: $((\gamma \to \alpha) \to (\alpha \to (\beta \to \gamma)))$

Termeni și tipuri:

Notăm $M : \sigma$ faptul că M are tipul σ.

- **Variabilă**: $x : \sigma$
- **Aplicare**: Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $MN : \tau$.
- **Abstractizare**: Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.M : \sigma \to \tau$.

Exemplu: Dacă $x : \sigma$, atunci funcția identitare are tipul $\lambda x.x : \sigma \to \sigma$.

- Termenul $x\ x$ nu poate avea niciun tip.

Church-typing vs Curry-typing

Asociere explicită (Church-typing):

- constă în prescrierea unui unic tip pentru fiecare variabilă, la introducerea acestuia
- presupune că tipurile variabilelor sunt explicit stabilite
- tipurile termenilor mai complecși se obțin natural, ținând cont de convențiile pentru aplicare și abstractizare

Asociere implicită (Curry-typing):

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

Exemplu: Pentru expresia $(\lambda zu.z)(y\ x)$, obținem prin:

- **Church-typing**: $x : \alpha \to \alpha, y : (\alpha \to \alpha) \to \beta, z : \beta, u : \gamma, \qquad M : \gamma \to \beta$
- **Curry-typing**: $x : \alpha, y : \alpha \to \alpha \to \beta, z : \alpha \to \beta, u : \alpha \to \alpha, \qquad M : (\alpha \to \alpha) \to \alpha \to \beta$

Sistem de deducție pentru Church $\lambda \to$

Mulțimea λ-termenilor cu pre-tipuri Λ_T este $\Lambda_T = x \mid \Lambda_T\Lambda_T \mid \lambda x : T.\Lambda_T$.

O afirmație este o expresie de forma $M : \sigma$, unde $M \in \Lambda_T$ și $\sigma \in T$; M - subiect; T - 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, iar $M : \sigma$ afirmație.

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

- $\Gamma \vdash x : \sigma$ (var), dacă $x : \sigma$
- Din $\Gamma \vdash M : \sigma \rightarrow \tau$ și $\Gamma \vdash N : \sigma$ rezultă $\Gamma \vdash MN : \tau$. $(\rightarrow_i)$
- Din $\Gamma, x : \sigma \vdash M : \tau$ rezultă $\Gamma \vdash (\lambda x : \sigma . M) : \sigma \rightarrow \tau$. $(\rightarrow_e)$

Un termen M este legal dacă $\Gamma \vdash M : \rho$.

Probleme decidabile pentru calculul Church $\lambda \rightarrow$:

- **Type checking:** verificarea că putem găsi o derivare pentru $context \vdash term : type$
- **Well-typedness (Typability):** verificarea că un termen e legal. Trebuie să găsim un context și un tip dacă termenul este legal, altfel să arătăm de ce nu se poate. $? \vdash term : ?$
- **Term finding:** dându-se un context și un tip, să stabilim dacă există un termen cu acel tip, în contextul dat: $context \vdash ? : type$

Alte tipuri

Tipul unit:

- Mulțimea tipurilor: $T = V \mid T \rightarrow T \mid Unit$
- Mulțimea λ-tipurilor cu pre-tipuri $\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T . \Lambda_T \mid unit$
- $\Lambda \vdash unit : Unit$

Tipul Void:

- Mulțimea tipurilor: $T = V \mid T \rightarrow T \mid Void$
- Mulțimea λ-tipurilor cu pre-tipuri $\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T . \Lambda_T \mid unit$
- Nu există regulă de tipuri deoarece tipul Void nu are inhabitant.

Tipul produs și constructorul pereche:

- Mulțimea tipurilor: $T = V \mid T \rightarrow T \mid Unit \mid Void \mid T \times T$
- Mulțimea λ-tipurilor cu pre-tipuri $\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T . \Lambda_T \mid unit \mid \langle \Lambda_T, \Lambda_T \rangle$

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

- Mulțimea tipurilor: $T = V \mid T \rightarrow T \mid Unit \mid Void \mid T \times T$
- Mulțimea λ-tipurilor cu pre-tipuri $\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T . \Lambda_T \mid unit \mid \langle \Lambda_T, \Lambda_T \rangle \mid fst\ \Lambda_T \mid snd\ \Lambda_T$

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

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

Tipul sumă și constructorii Left/Right:

- Mulțimea tipurilor: $T = V \mid T \to T \mid Unit \mid Void \mid T \times T \mid T + T$
- Mulțimea λ-tipurilor cu pre-tipuri $\Lambda_T = x \mid \Lambda_T \Lambda_T \mid \lambda x : T.\Lambda_T \mid unit \mid \langle \Lambda_T, \Lambda_T \rangle$ $fst\ \Lambda_T \mid snd\ \Lambda_T \mid Left\ \Lambda_T \mid Right\ \Lambda_T \mid case\ \Lambda_T\ of\ \Lambda_T;\ \Lambda_T$

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

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

Corespondența Curry-Howard

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

Logica intuiționistă

- Logică constructivistă
- Bazată pe noțiunea de demonstrație

- 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)

FLP Cheat Sheet

Lambda calcul - elemente de bază

Funcții

Fie $f,\ g : X \to Y$, $f(x) = x^2 - 1, g(x) = (x-1)(x+1)$

extensional egale: $f(x) = g(x), \forall x \in X.$ (DA)

intensional egale: sunt definite de aceeași formulă. (NU)

Termeni lambda

M, N ::= x (*variabilă*)
| (MN) (*aplicare*)
| $(\lambda x.M)$ (*abstractizare*)

Convenții

1. aplicarea e asociativă la stânga:
$$f\ x\ y\ z = ((f\ x)\ y)\ z$$
2. corpul abstractizării se extinde la dreapta:
$$\lambda x.M\ N = \lambda x.(M\ N)$$
3. mai mulți λ se comprimă:
$$\lambda xyz.M = \lambda x.\lambda y.\lambda z.M$$

Variabile

Exemplu: $\lambda x.N$

1. operator de legare(binder): $\lambda\_.\_$
2. variabilă de legare(binding): x
3. domeniu de legare a lui x: N
4. termen fără variabile libere: închis/combinator

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

1. Mulțimea variabilelor legate: $\{x, y\}$
2. Mulțimea variabilelor libere *(FV(M))*: $\{y, z\}$

Obs. $FV(x) = x$
$FV(M\ N) = FV(M) \cup FV(N)$
$FV(\lambda x.M) = FV(M) \setminus x$

α-echivalența

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

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

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

Substituții

[*Variabilă*] (x[u/x] = u) sau (y[u/x] = y, dacă x ≠ y)
[*Aplicare*] (M N)[u/x] = (M[u/x] N[u/x])
[*Abstractizare*]

$$\lambda y.t[u/x] = \begin{cases} \lambda y.(t[u/x]),\ y \neq x,\ y \notin FV(u) \\ \lambda y'.(t\langle y'/y\rangle[u/x]),\ y \neq x,\ y \in FV(u) \end{cases}$$

Lambda calcul - β-reducții

- β-redex: termen de forma $(\lambda x.M)\ N$
- Redusul redex-ului $(\lambda x.M)N : M[N/x]$
- Forma normală: un lambda termen fără redex-uri
- Daca $\exists$ N în forma normală a.î. M $\twoheadrightarrow_\beta$ N $\Rightarrow$ M e slab normalizabil.
- Dacă $\nexists$ reduceri infinite care încep din M $\Rightarrow$ M e puternic normalizabil.
- Algoritm: reducem lambda termeni prin găsirea unui subtermen care e redex, îl înlocuim cu redusul său și ciclăm până nu mai sunt redex-uri.

β-reducții

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

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

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

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

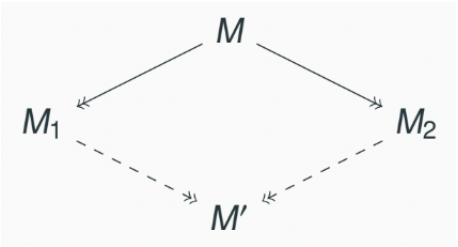
Exemplu:

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

Observații:
- reducerea unui redex poate crea/șterge redex-uri
- găsirea unei forme normale depinde de ordinea reducerii redex-urilor

Teorema Church-Rosser

Dacă M $\twoheadrightarrow_\beta$ M1 și M $\twoheadrightarrow_\beta$ M2 atunci există M' astfel încât M1 $\twoheadrightarrow_\beta$ M' și M2 $\twoheadrightarrow_\beta$ M'.



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

Strategii de evaluare

Strategia normală: leftmost-outermost

M1, M2 redex-*uri*
M1 = subtermen M2
$\Big\}\to$ M1 nu e viitorul redex ales

Strategia aplicativă: leftmost-innermost

M1, M2 redex-*uri*
M1 = subtermen M2
$\Big\}\to$ M2 nu e viitorul redex ales

Strategia call-by-name - strategia normală fără a face reduceri în corpul unei $\lambda - abstractizări$.

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

Strategia call-by-value - strategia aplicativă fără a face reduceri în corpul unei $\lambda - abstractizări$.

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

Expresivitatea λ-calculului

Booleeni

$T \triangleq \lambda xy.x$ if $\triangleq \lambda btf.b\ t\ f$ or $\triangleq \lambda xy.if\ x\ T\ y$
$F \triangleq \lambda xy.y$ not $\triangleq \lambda x.if\ x\ F\ T$ and $\triangleq \lambda xy.if\ x\ y\ F$

Numere naturale

Numeralul Church $\bar{n} \triangleq \lambda fx.f^n x$
Succ $\triangleq \lambda nfx.f\ (n\ f\ x) \implies$ Succ $\bar{n} = \overline{n+1}$
add $\triangleq \lambda mn.m$ Succ n mul $\triangleq \lambda mn.m$ (add n)
exp $\triangleq \lambda mn.m$ (mul n) isZero $\triangleq \lambda nxy.n\ (\lambda z.y)\ x$

Puncte fixe

F, M sunt λ-termeni
F M $=_\beta$ M
$\Big\}\to$ M este punct fix al lui F

În λ-calcul fără tipuri, orice termen are punct fix.
Combinator de puncte fixe = termen închis care construiește un punct fix pe un termen arbitrar.
- Curry: $Y \triangleq \lambda y.(\lambda x.y\ (x\ x))\ (\lambda x.y\ (x\ x))$
- Turing: $\Theta \triangleq (\lambda xy.y\ (x\ x\ y))\ (\lambda xy.y\ (x\ x\ y))$

fact $\triangleq Y\ F$ [Y F e punct fix pentru F]
fact $\triangleq Y(\lambda fn.if\ (isZero\ n)\ (\bar{1})\ (mul\ n\ (f(pred\ n))))$

FLP Cheat Sheet

Lambda calcul cu tipuri simple

Mulțimea tuturor tipurilor simple T este definită prin T = V | T → T, unde V = {α, β, γ, ...} este o mulțime infinită de tipuri variabilă.

Exemplu:
- α
- $((\gamma \to \alpha) \to (\alpha \to (\beta \to \gamma)))$

[**Tipul variabilă**] Dacă $\alpha \in$ V, atunci $\alpha \in$ T.
[**Tipul săgeată**] Dacă $\delta, \tau \in$ T, atunci $(\delta \to \tau) \in$ T.
[paranteze asociative la dreapta]

[**Variabilă**] x : δ.
[**Aplicare**] Dacă M : $\delta \to \tau$ și N : $\delta \Rightarrow M\ N : \tau$.
[**Abstractizare**] Dacă x : δ, M : $\tau \Rightarrow \lambda$x.M : $\delta \to \tau$.

Dacă $\exists$ un tip δ a.î. M : $\delta \Rightarrow$ M are **tip** (e **typeable**).

Convenții
1. $y\ x$ poate avea un tip doar dacă y are un tip săgeată de forma $\delta \to \tau$ și tipul lui x se potrivește cu tipul domeniu δ. Astfel, $y\ x : \tau$.
2. Termenul $x\ x$ nu poate avea nici un tip.
   a. apariția I: x : $\delta \to \tau$.
   b. aparația II: x : δ.
   Cum orice variabilă are un unic tip, ar trebui ca $\delta \to \tau \equiv \delta$, ceea ce este imposibil.

Lambda calcul - tipuri

Lambda calcul fără tipuri
nu se specifică tipul niciunei expresii
nu se specifică domeniul/codomeniul funcțiilor

Lambda calcul cu tipuri simple
se specifică mereu tipul oricărei expresii
nu se poate aplica o funcție unui argument care are alt tip față de domeniul funcției
expresiile de forma f(f) sunt eliminate

Lambda calcul cu tipuri polimorfice
se poate specifica că o expresie are tipul X → X, fără a specifica cine de fapt este X

Sistem de deducție pentru Church $\lambda\to$

Mulțimea λ-termenilor cu pre-tipuri Λ_T :

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

- O **afirmație** este o expresie de forma M : δ, unde $M \in \Lambda_T$ și $\delta \in$ T.
- În această afirmație, M se numește **subiect**.
- O **declarație** este o afirmație de forma x : δ.
- Un **context** Γ este o listă de declarații cu subiecți diferiți.
- O **judecată** este o expresie de forma $\Gamma \vdash$ M : δ.

Sistem de deducție pentru Church $\lambda\to$

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

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

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

Exemplu:
1) y : $\alpha \to \beta$
2) z : α
$\lambda y.\lambda z.yz$ are tipul $(\alpha \to \beta) \to \alpha \to \beta$ în contextul vid.

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

Probleme decidabile

Type-checking: se verifică posibilitatea de găsire a unei derivări pentru [$context \vdash term : type$].
Well-typedness (Typability): se verifică dacă un termen este legal [$? \vdash term : ?$].
Type Assignment: se găsește tipul când contextul este dat [$context \vdash term : ?$].
Term Finding: având un context și un tip anumit, se verifică dacă există un termen cu acel tip, în contextul dat. [$context \vdash ? : type$].

Alte tipuri

Mulțimea tipurilor
T = V | T → T | *Unit* | *Void* | T x T | T + T
Mulțimea λ-termenilor cu pre-tipuri Λ_T :

$$\Lambda_T = x \mid \Lambda_T\ \Lambda_T \mid \lambda x : \mathrm{T}.\Lambda_T \mid unit \mid <\Lambda_T, \Lambda_T> \mid fst\Lambda_T \mid snd\ \Lambda_T \mid Left\ \Lambda_T \mid Right\ \Lambda_T \mid case\ \Lambda_T\ of\ \Lambda_T;\ \Lambda_T$$

Teoria tipurilor	**Logica**
tipuri	formule
termeni	demonstrații
inhabitation a tipului δ	demonstrație a lui δ
tipul produs	conjuncție
tipul funcție	implicație
tipul sumă	disjuncție
tipul void	false
tipul unit	true

Church-typing

Tip unic explicit stabilit pentru fiecare variabilă

Exemplu: Tipul expresiei (λzu.z) (y x) - ?, știind că:
1)x : $\alpha \to \alpha$ 2)y : $(\alpha \to \alpha) \to \beta$ 3)z : β 4)u : γ

- Aplicare (1) și (2) $\Rightarrow$ (5): y x : β.
- Abstractizare (4) și (3) $\Rightarrow$ (6): λu.z : $\gamma \to \beta$.
- Abstractizare (3) și (6) $\Rightarrow$ (7): λzu.z : $\beta \to \gamma \to \beta$.
- Aplicare (7) și (5) $\Rightarrow$ (λzu.z) (y x) : $\gamma \to \beta$.

Curry-typing

Nu se prescrie un tip pentru fiecare variabilă

Exemplu: Tipul expresiei M = (λzu.z) (y x)

- M e o aplicare $\Rightarrow \lambda$zu.z : A → B, $y\ x$: A, M : B.
- λzu.z : A → B $\Rightarrow$ z : A și λu.z : B.
- B e tipul unei abstractizări $\Rightarrow$ u : C și z : D.
- y x e o aplicare $\Rightarrow$ y : E → F, x : E și y x : F.
- z : A și z : D, y x : A și y x : F $\Rightarrow$ A ≡ F.

Astfel se obține schema generală:
x : E y : E → A z : A u : C M : C → A

Se pot considera și tipuri reale în schema de mai sus:
x : β y : $\beta \to \alpha$ z : α u : δ M : $\delta \to \alpha$

Lambda Calcul

Lambda termeni

lambda termen = variabila
| aplicare
| abstracticare

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

Conventii:

- Se elimina parantezele exterioare
  (M N) P inseamna M N P
- Aplicarea este asociativa la stanga
  - M N P inseamna (M N) P
  - f x y z inseamna ((f x) y) z
- Corpul abstractizarii se extinde la dreapta
  $\lambda x.MN$ inseamna $\lambda x.(MN)$
- In termenul $M \equiv (\lambda x.xy)(\lambda x.yz)$
  - λ se numeste operator de legare
  - x este variabila legata
  - z este variabila libera
  - y are o aparitie legata, si una libera
  - multlimea variabilelor libere ale lui M este y, z
- N din $\lambda x.N$ se numeste **domeniul** de legare a lui x si toate aparitiile lui x in N sunt **legate**
- un termen fara variabile libere se numeste **inchis**(closed) sau **combinator**

$\beta - reductii$ reguli

Un pas de $\beta - reductie$ $\rightarrow_\beta$ este cea mai mica relatie pe lamba termeni care satisface regulile:

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

$$\frac{M \rightarrow_\beta M'}{MN \rightarrow_\beta M'N}$$

$$\frac{N \rightarrow_\beta N'}{MN \rightarrow_\beta MN'}$$

$$\frac{M \rightarrow_\beta N'}{\lambda x.M \rightarrow_\beta \lambda x.M'}$$

Redenumire de variabile

M $\langle y/x \rangle$ - rezultatul obtinut dupa redenumirea lui x cu y in M

- $x\langle y/x\rangle \equiv y$
- $z\langle y/x\rangle \equiv z$, daca $x \neq z$
- $(MN)\langle y/x\rangle \equiv (M\langle y/x\rangle)(N\langle y/x\rangle)$
- $(\lambda x.M)\langle y/x\rangle \equiv \lambda y.(M\langle y/x\rangle)$
- $(\lambda z.M)\langle y/z\rangle \equiv \lambda z.(M\langle y/x\rangle)$, daca $x \neq z$

$\alpha - echivalenta$

$\alpha - echivalenta$ - cea mai mica relatie de congruenta pe multimea lambda termenilor, astfel incat pentru orice termen M si orice variabila y care nu apare in M, avem: $\lambda.M =_\alpha \lambda y.(M\langle y/x\rangle)$

Substitutii

$M[N/x]$ este rezultatul obtinut dupa inlocuirea lui x cu N in M.
1. Vrem sa inlocuim doar variabile libere:
$x(\lambda xy.x)[N/x]$ este $N(\lambda xy.x)$ (NU $N(\lambda xy.N)$ sau $N(\lambda Ny.N)$
2. Nu vream sa legam variabile libere neintentionat asa ca redenumim variabilele legate inainte de substitutie.

$M[N/x]$

$x[N/x] \equiv N$
$y[N/x] \equiv y$, daca $x \neq y$
$(MP)[N/x] \equiv (M[N/x])(P[N/x])$
$(\lambda x.M)[N/x] \equiv \lambda x.M$
$(\lambda y.M)[N/x] \equiv \lambda y.(M[N/x])$, $x \neq y, y \notin FV(N)$
$(\lambda y.M)[N/x] \equiv \lambda y'.(M\langle y'/y\rangle[N/x])$, daca $x \neq y$, $y \in FV(N)$, y' variabila noua
$FV(N)$ - multimea variabilelor libere ale lui N

$\beta - reductii$

$\beta - reductie$ = procesul de a evalua lambda termeni prin "pasarea de argumente"
$\beta - redex$ = un termen de forma $(\lambda x.M)N$
redusul unui redex $(\lambda x.M)N$ este $M[N/x]$
forma normala = un lambda termen fara redex-uri

$\beta - forma$ normala

$M \twoheadrightarrow_{beta}$ - M poate fi $\beta - redus$ pana la M' in 0 sau mai multi pasi
M este:
slab normalizabil daca exista N in forma normala a.i. $M \twoheadrightarrow_\beta N$.
puternic normalizabil daca nu exista reduceri infinite care incep din M.
$(\lambda xy.y)((\lambda x.xx)(\lambda x.xx))(\lambda z.z)$ este **slab normalizabil**, dar NU puternic normalizabil.

Teorema Church-Rosser

Daca $M \twoheadrightarrow_\beta M_1$ si $M \twoheadrightarrow_\beta M_2$ atunci exista M' a.i. $M_1 \twoheadrightarrow_\beta M'$ si $M_2 \twoheadrightarrow_\beta M'$.
Consecinta. Un lambda termen are cel mult o $\beta - forma$ normala (modulo $\alpha - echivalenta$).

Strategii de evaluare

Strategia normala(leftmost-outermost)

Daca M_1 si M_2 sunt redex-uri si M_1 este un subtermen al lui M_2, atunci M_1 NU va fi urmatorul redex ales.
$\underline{(\lambda xy.y)((\lambda x.xx)(\lambda x.xx))}(\lambda z.z) \rightarrow_\beta \underline{(\lambda y.y)(\lambda x.x)}$
$\rightarrow_\beta \lambda x.x$

Strategia aplicativa(leftmost-innermost)

Daca M_1 si M_2 sunt redex-uri si M_1 este un subtermen al lui M_2, atunci M_2 NU va fi urmatorul redex ales.
$(\lambda xy.y)\underline{((\lambda x.xx)(\lambda x.xx))}(\lambda z.z)$
$\rightarrow_\beta (\lambda xy.y)((\lambda x.xx)(\lambda x.xx))(\lambda z.z)$

CBN si CBV

Strategia call-by-name(CBN) = strategia normala fara a face reducrei in corpul unei $\lambda - abstractizari$
Strategia call-by-value(CBV) = strategia aplicativa fara a face reduceri in corpui unei $\lambda - abstractizari$

Expresivitatea λ - calculului

Booleeni

T $\triangleq \lambda xy.x$
F $\triangleq \lambda xy.y$
if $\triangleq \lambda btf.$b t f
and $\triangleq \lambda b_1 \lambda b_2$ **if** $b_1 b_2$**F**
or $\triangleq \lambda b_1 \lambda b_2$ **if** b_1**T**b_2
not $\triangleq \lambda b_1$ **if** b_1**FT**

Numere naturale

Numeralii Church: $\lambda fx.f^n x$
$\bar{0} \triangleq \lambda fx.f^0 x = \lambda fx.x$
$\bar{1} \triangleq \lambda fx.f^1 x = \lambda fx.fx$
$\bar{2} \triangleq \lambda fx.f^1 x = \lambda fx.f(fx)$
Succesor: **Succ** $\triangleq \lambda nfx.f(nfx)$
Adunare:
$add \triangleq \lambda mnfx.mf(nfx)$
$add \triangleq \lambda mn.m$ **Succ** n
Inmultirea: **mul** $\triangleq \lambda mn.m($**add**$n)\bar{0}$
Ridicarea la putere: **exp** $\triangleq \lambda mn.m($**mul**$n)\bar{1}$
Verifica numar $= 0$: **isZero** $\triangleq \lambda nxy.n(\lambda z.y)x$

Puncte fixe

Spunem ca x este un punct fix al functiei f daca $f(x) = x$
THM. In lambda calcul fara tipuri, orice termen are un punct fix.
Daca F si M sunt λ-termeni, spunem ca M este un **punct fix** al lui F daca $FM =_\beta M$

Combinatorii de puncte fixe:
Curry: $\mathbf{Y} \triangleq y.(\lambda x.y(xx))(\lambda x.y(xx))$
Turing: $\mathbf{\Theta} \triangleq (\lambda xy.y(xxy))(\lambda xy.y(xxy))$

Factorial

fact n = if(**isZero** $n)(\bar{1})($**mul** $n($**fact**$($**pred** $n)))$
fact $\triangleq \mathbf{Y}(fn.$if (**isZero** $n)(\bar{1})($**mul** $n(f($**pred** $n)))$

Lambda calcul cu tipuri

Tipuri simple

Fie $\mathbb{V} = \{\alpha, \beta, \gamma, ...\}$ o multime infinita de tipuri variabila. Multimea tuturor tipurilor simple $\mathbb{T}$ este definita prin:

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

(Tipul variabila) Daca $\alpha \in \mathbb{V}$, atunci $\alpha \in \mathbb{T}$
(Tipul sageata) Daca $\sigma, \tau \in \mathbb{T}$, atunci $(\sigma \to \tau) \in \mathbb{T}$

Termeni si tipuri

Variabila. $x : \sigma$
Aplicare. Daca $M : \sigma \to \tau$ si $N : \sigma$, atunci $MN : \tau$.
Abstractizare. Daca $x : \sigma$ si $M : \tau$, atunci $\lambda x.M : \sigma \to \tau$

Deductie pentru Church $\lambda \to$

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

$$\Lambda_\mathbb{T} = x | \Lambda_\mathbb{T} \Lambda_\mathbb{T} | \lambda x : \mathbb{T}.\lambda_\mathbb{T}$$

O afirmatie este o expresie de form $M : \sigma$, unde $M \in \Lambda_\mathbb{T}$ si $\sigma \in \mathbb{T}$ (M- subiect, σ- tip)
O declaratie este o afirmatie in care subiectul este o variabila $(x : \sigma)$
Un context este o lista de declaratii cu subiecti diferiti.
O judecata este o expresie de forma $\Gamma \vdash M : \sigma$, unde $\vdash$ este context si $M : \sigma$ este o afirmatie.

Reguli

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

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

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

Strategii de evaluare

Multimea tipurilor:

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

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

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

$|\text{fst } \Lambda_\mathbb{T} |\text{snd } \Lambda_\mathbb{T} |\text{Left } \Lambda_\mathbb{T} |\text{Right } \Lambda_\mathbb{T} |\text{case } \Lambda_\mathbb{T} \text{of } \Lambda_\mathbb{T}; \Lambda_\mathbb{T}$

· Corespondenta Curry-Howard ·

λ - calcul cu tipuri

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

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

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

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

$$\frac{\Gamma \vdash \lambda x.M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\to_E)$$

Deductie naturala

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

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

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

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

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

Teoria tipurilor $\to$ *Logica*
tipuri $\to$ *formule*
termeni $\to$ *demonstratii*
inhabitation a tipului $\sigma \to$ demonstratie a lui σ
tip produs $\to$ conjunctie
tip functie $\to$ implicatie
tip suma $\to$ disjunctie
tipul void $\to$ false
tipul unit $\to$ true
Propositions are types!
Proofs are terms!

Template folosit: Nelson Elias Rincon Quiñones

Cheatsheet FLP

Litere grecești:

$A\alpha$ $alfa$	$B\beta$ $beta$	$\Gamma\gamma$ $gamma$	$\Delta\delta$ $delta$	$E\epsilon$ $epsilon$	$Z\zeta$ $zeta$	$H\eta$ eta	$\Theta\theta$ $theta$	$I\iota$ $iota$	$K\kappa$ $kappa$	$\Lambda\lambda$ $lambda$	$M\mu$ mu
$N\nu$ nu	$\Xi\xi$ xi	Oo $omicron$	$\Pi\pi$ pi	$P\rho$ rho	$\Sigma\sigma$ $sigma$	$T\tau$ tau	$\Upsilon\upsilon$ $upsilon$	$\Phi\phi$ phi	$X\chi$ chi	$\Psi\psi$ psi	$\Omega\omega$ $omega$

Lambda termen = variabilă | aplicare | abstractizare

$$M ::= x|(MM)|(\lambda x.M)$$

Convenții:

- $MNP = (MN)P$, $fxyz = ((fx)y)z$ (asociativitate la stânga a aplicării)
- $\lambda x.MN = \lambda x.(MN)$, deci $\lambda x.MN \neq (\lambda x.M)N$ (extindere la dreapta a corpului abstractizării)
- $\lambda xyz.M = \lambda x.\lambda y.\lambda z.M$ (comprimarea abstractizărilor)

Pentru $\lambda x.M$ avem:

- λ = operator de legare (*binder*)
- x = variabilă de legare (*binding*)
- N = domeniul (*scope*) de legare al lui x
- Aparițiile lui x în N sunt *legate*
- O apariție ce nu este legată se numește *liberă*
- Un termen fără variabile libere se numește *închis*, iar un termen închis se mai numește și *combinator*

$FV(M)$ **- mulțimea variabilelor libere**

$$FV(x) = \{x\}$$
$$FV(M) = FV(M) \cup FV(N)$$
$$FV(\lambda x.M) = FV(M) \backslash \{x\}$$

$M < y/x >$ reprezintă rezultatul obținut după redenumirea variabilei x cu y în lambda termenul M

α**-echivalență** = egalitate între doi termeni (modulo redenumire de variabile legate), adică

$$\lambda x.M =_\alpha \lambda y.(M < y/x >)$$

$M[N/x]$ este rezultatul obținut după înlocuirea lui x cu N în M și are formula generală:

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

Convenție: $M = N \Leftrightarrow M =_\alpha N$ (2 termeni sunt egali dacă sunt α echivalenți)

β-reducție = procesul de a evalua lambda termeni prin *pasarea de argumente funcțiilor*
β-redex = termen de forma $(\lambda x.M)N$
Redusul unui redex $(\lambda x.M)N$ este $M[N/x]$
Lambda termenii se reduc prin găsirea unui redex căruia i se aplică β-reducția
Forma normală = lambda termen fără redex-uri

Un pas de β-reducție($\rightarrow_\beta$) = cea mai mică relație de lambda termeni ce satisface regulile:

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

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

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

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

Exemplu β-**reducție**:

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

Observație: Există lambda termeni care nu pot fi reduși la o β-formă normală deoarece evaluarea este infinită (ex: $(\lambda x.xx)(\lambda x.xx)$) sau care pot fi reduși, dar pot să nu o atingă niciodată (alegerea β-redexurilor duce la un lambda termen ireductibil)

$M \twoheadrightarrow_\beta M' \Leftrightarrow M$ poate fi β-redus până în M' în 0 sau mai mulți pași
M-slab normalizabil $\Leftrightarrow (\exists)N$-formă normală a.i. $M \twoheadrightarrow_\beta N$
M-puternic normalizabil $\Leftrightarrow (\nexists)$ reduceri infinite care încep din M
Orice termen puternic normalizabil este și slab normalizabil.

Teorema Church-Rosser: Dacă $M \twoheadrightarrow_\beta M_1$ și $M \twoheadrightarrow_\beta M_2 \Rightarrow \exists M'$ a.i. $M_1 \twoheadrightarrow_\beta M'$ și $M_2 \twoheadrightarrow_\beta M'$.
Consecință: Un lambda termen are cel mult o β-formă normală (modulo α-echivalență).

Strategii de evaluare:

1. Strategia normală (*leftmost-outermost*)
   Dacă M_1 M_2 sunt redex-uri și M_1-subtermen al lui $M_2 \Rightarrow M_1$ nu va fi următorul redex ales.
   $$\underline{(\lambda xy.x)((\lambda x.xx)(\lambda x.xx))}(\lambda z.z)$$
2. Strategia aplicativă (*leftmost-innermost*)
   Dacă M_1 M_2 sunt redex-uri și M_1-subtermen al lui $M_2 \Rightarrow M_2$ nu va fi următorul redex ales.
   $$(\lambda xy.x)(\underline{(\lambda x.xx)(\lambda x.xx)})(\lambda z.z)$$

Strategii în programarea funcțională:

1. Call-by-name (CBN) = strategia **normală** fără a face reduceri în corpul unei λ-abstractizări
2. Call-by-value (CBV) = strategia **aplicativă** fără a face reduceri în corpul unei λ-abstractizări (strategie folosită în Haskell = *lazy evaluation*)

Valoare = λ-termen pentru care nu există β-reducții pe strategia de evaluare considerată

$\lambda x.x$ - mereu valoare
$(\lambda x.x)1$ - nu mereu valoare

Valori boolene:

b - valoare booleană
x, y - λ-termeni oarecare

$$T \triangleq \lambda xy.x \qquad F \triangleq \lambda xy.y$$

$$\textbf{if} = \lambda bxy. \begin{cases} x, \textbf{if } b = true \\ y, \textbf{if } b = false \end{cases}$$

$$\textbf{and} \triangleq \lambda b_1 b_2.\textbf{if } b_1 b_2 F$$
$$\textbf{or} \triangleq \lambda b_1 b_2.\textbf{if } b_1 T b_2$$
$$\textbf{not} \triangleq \lambda b_1 b_2.\textbf{if } b_1 F T$$

Numerali Church:

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

$\textbf{Succ} \triangleq \lambda nfx.f(nfx)$
$\textbf{add} \triangleq \lambda mnfx.mf(nfx) \triangleq \lambda mn.m\textbf{Succ}\ n$

Exemplu:

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

$\textbf{mul} \triangleq \lambda mn.m(\textbf{add}\ n)\ \overline{0}$
$\textbf{exp} \triangleq \lambda mn.m(\textbf{mul}\ n)\ \overline{1}$
$\textbf{isZero} \triangleq \lambda nxy.n(\lambda z.y)x$

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

- $\twoheadrightarrow_\beta$ este închiderea reflexivă și tranzitivă a relației $\rightarrow_\beta$
- $=_\beta$ este închiderea reflexivă, simetrică și tranzitivă a relației $\rightarrow_\beta$

Exemplu:

$$(\lambda y.yv)z =_\beta (\lambda x.zx)v$$
deoarece
$$(\lambda y.yv)z \rightarrow_\beta zv \leftarrow_\beta (\lambda x.zx)v$$

Punct fix al unui lambda termen F este un lambda termen M care verifică $FM =_\beta M$.

Teoremă: În lambda calcul fără tipuri orice termen are un punct fix.

Combinatori de punct fix:

- Combinatorul de punct fix al lui Curry
$$\mathbf{Y} \triangleq \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$$
- Combinatorul de punct fix al lui Turing
$$\Theta \triangleq (\lambda xy.y(xxy))(\lambda xy.y(xxy))$$

$\textbf{fact} \triangleq \mathbf{Y}\ (\lambda fn.\textbf{if}\ (\textbf{isZero}\ n)(\overline{1})(\textbf{mul}\ n(f(\textbf{pred}\ n))))$

Lambda calcul cu tipuri:

$\mathbb{V} = \{\alpha, \beta, \gamma, \dots\}$ este o mulțime finită de tipuri variabilă.
$\mathbb{T} = \underbrace{\mathbb{V}}_{\text{variabilă}} \mid \underbrace{\mathbb{T} \to \mathbb{T}}_{\text{săgeată}}$ este mulțimea tuturor tipurilor simple.

Exemple de tipuri simple:

- γ - variabilă
- $(\beta \to \gamma)$ - săgeată (funcție)
- $(\gamma \to \gamma) \to (\alpha \to (\beta \to \gamma))$ - săgeată (funcție)

Asociativitate:

- $\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4 \equiv (\alpha_1 \to (\alpha_2 \to (\alpha_3 \to \alpha_4)))$ (asociativitate la dreapta a tipului săgeată)
- $x_1x_2x_3x_4 \equiv (((x_1x_2)x_3)x_4)$ (asociativitate la stânga a tipului variabilă)

$M : \sigma$ înseamnă că M are tipul σ.
O variabilă x dintr-un termen M are un unic tip. (i.e dacă $x : \sigma$ și $x : \tau$ atunci $\sigma \equiv \tau$)
Dacă $M : \sigma \to \tau$ și $N : \sigma$ atunci $MN : \tau$
Dacă $M : \tau$ și $x : \sigma$ atunci $\lambda x.M : \sigma \to \tau$
M este *typeable* (are tip) dacă $(\exists)\ \sigma$ a.i. $M : \sigma$

Metode de asociere de tipuri variabilelor:

- Church-typing (asociere explicită). Fiecare variabilă are un tip la introducerea ei.
- Curry-typing (asociere implicită). Nu se prescrie nici un tip variabilelor la început, iar termenii typeable sunt descoperiți printr-un proces de căutare ce poate presupune o "ghicire"de tipuri.

Tipul variabilelor libere este dat de un *context*. În lambda calcul cu tipuri, termenul $(\lambda z.\lambda u.z)(yx)$ se scrie:

$$(\lambda z : \beta.\lambda u : \gamma.z)(yx)$$

Dacă presupunem un context pentru variabilele libere știute, scriem:

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

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

→ **Afirmație** - expreesie de forma $M : \sigma$, unde $M \in \Lambda_{\mathbb{T}}$ și $\sigma \in \mathbb{T}$ (M se numește *subiect*, și σ *tip*)

→ **Declarație** - afirmație în care subiectul este variabilă

→ **Context** - listă de declarații cu subiecți diferiți

→ **Judecată** - expresie de forma $\Gamma \vdash M : \sigma$, unde Γ este context și $M : \sigma$ este afirmație

Reguli pentru stabilirea unui tip într-un anumit context:

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

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

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

M este termen *legal* dacă există un context Γ și un tip ρ astfel încât $\Gamma \vdash M : \rho$
Exemplu: Arătăm că termenul $\lambda y : \alpha \to \beta.\lambda z : \alpha.yz$ are tipul $(\alpha \to \beta) \to \alpha \to \beta$ în contextul vid.

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

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

Probleme de rezolvat în teoria tipurilor:

1. **Type Checking**
   Verificarea dacă există o derivare pentru
   $$\text{context} \vdash \text{term} : \text{type}$$
2. (a) **Well-typedness (typability)**
   Verificarea dacă un termen este legal (i.e găsirea unui context și a unui tip dacă termenul e legal, altfel să arătăm de ce nu se poate)
   $$? \vdash \text{term} : ?$$
   (b) **Type Assignment**
   Contextul este dat și trebuie găsit tipul
   $$\text{context} \vdash \text{term} : ?$$
3. **Term Finding (Inhabitation)**
   Se cunosc contextul și tipul, trebuie să stabilim dacă există un termen cu acel tip în contextul dat
   $$\text{context} \vdash ? : \text{type}$$

Limitări are λ-calculului cu tipuri simple:

- Nu mai avem recursie nelimitată deoarece combinatorii de punct fix nu sunt *typeable*.
- Tipurile pot fi prea restrictive. $(\lambda f.\mathbf{if}(fT)(f3)(f5))(\lambda x.x))$

Soluții posibile:

- Let-polymorphism - variabilele libere se redenumesc la fiecare folosire
  $$\mathbf{let}\ f = \lambda x.x\ \mathbf{in}$$
  $$\mathbf{if}\ (fT)(f3)(f5)$$
- Cuantificatori de tipuri
  $$\lambda x.x : \Pi\alpha.\alpha \to \alpha$$
  Operatorul de legare Π face ca variabila de tip α să nu fie rigidă.

Tipul **Unit**:

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \mathbf{Unit}$$
$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\Lambda_{\mathbb{T}} \mid \mathbf{unit}$$

$$\overline{\Gamma \vdash \mathbf{unit}{:}\mathbf{Unit}\,(unit)}$$

Tipul **Void**:

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \mathbf{Unit} \mid \mathbf{Void}$$
$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\Lambda_{\mathbb{T}} \mid \mathbf{unit}$$

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

Tipul **produs** și **constructorul pereche**:

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid \mathbf{Unit} \mid \mathbf{Void} \mid \mathbb{T} \times \mathbb{T}$$
$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\Lambda_{\mathbb{T}} \mid \mathbf{unit} \mid <\Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}}> \mid fst\ \Lambda_{\mathbb{T}} \mid snd\ \Lambda_{\mathbb{T}}$$

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

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

Tipul **Sumă** și **constructorii Left/Right**:

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

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

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

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

Propositions
are types!
♡

Faptul că există un termen de tip σ (inhabitant de tip σ) înseamnă că σ este o teoremă și are o demonstrație în ⋆LOGICĂ⋆!
♡

FUNDAMENTELE LIMBAJELOR DE PROGRAMARE

Cursul 1

Functiile prin grafic
= cele clasice de la matematica (cu domeniu si codomeniu fixat + functie)
= definite extensional (singurul lucru pe care il observam e cum functia duce intrarile in iesiri)
Functiile ca reguli sau formule
= nu e mereu necesar sa stim domeniu si codomeniu
= definite astfel incat sa putem observa si alte detalii

Functii extensional egale	Functii intensional egale
pentru aceeasi intrare obtin aceeasi iesire	au aceeasi formula
$F(x) = G(x)$ pentru orice x din domeniu	$F(x) = x^2 + 1$ $G(x) = (x+1)(x-1)$

Lambda calculul = teoria functiilor ca formule
= sistem care permite manipularea functiilor ca expresii

Fie f functia $x \rightarrow x^2$ si A = f (5). In lambda calcul avem A = $(\lambda x.x^2)$ (5).
functia care duce x in x^2
x este locala/legata in termenul $\lambda x.x^2$

Functiile de nivel inalt = intrarile/iesirile lor sunt tot functii
f o f = λx.f (f (x))
f -> f o f = λf.λx.f (f (x))

Exemplu de evaluare a unei functii:
((λf .λx.f (f (x)))(λy.y2))(5) = 625

Lambda Calcul		
Fara tipuri	Cu tipuri simple	Cu tipuri polimorfice
Nu specificam tipul expresiei, domeniul sau codomeniul functiilor	Specificam mereu tipul oricarei expresii, iar expresiile de forma f(f) dispar	Specificam ca o expresie are tipul X -> X, dar nu specificam cine este X
Avem flexibilitate, dar putem avea erori cand aplicam o functie unui argument pe care nu il poate procesa	Nu avem flexibilitate, nu putem aplica functii unui argument care are alt tip fata de domeniul functiei	

Functie calculabila = exista o metoda pe foaie pentru a calcula f(n) pentru orice n

1. Turing = ddaca poate fi calculata de o masina Turing
2. Godel = ddaca este recursiva
3. Church = ddaca poate fi scrisa ca un lambda termen

Toate aceste trei modele de calculabilitate sunt echivalente.

Lambda calcul – Logica constructiva (o demonstratie trebuie sa fie o constructie, un program, iar lambda calculul este o notatie pentru astfel de programe)

- Logica clasica = plec de la presupuneri si ajung la contradictie
- Logica constructiva = ca sa arat ca un obiect exista, il construiesc explicit

Cursul 2

V = multime infinita de variabile (x, u, z ...)
Lambda termeni: lambda termen = variabila | aplicare | abstractizare

M, N :: = x | (MN) | (λx.M)

În Haskell, \ e folosit în locul simbolului si λ în locul punctului:

λx.x*x ----------> \x -> x * x
λx.x > 0 ----------> \x -> x > 0

Conventii:

1. Se elimina parantezele exterioare
2. Aplicarea este asociativa la stanga: MNP = (MN)P
3. Corpul abstractizarii se extinde la dreapta cat mai mult: λx.MN = λx.(MN)
4. Mai multi λ pot fi comprimati: λxyz.M = λx.λy.λz.M

Variabiile libere si variabile legate:

- λ_._ = operator de legare
- x din λx._ = variabila de legare
- N din λx.N = domeniu de legare a lui x

Toate aparitiile lui x in N sunt legate.
O aparitie cacre nu e legata = libera
Un termen fara variabile libere = inchis sau combinator

Cum gasim variabilele libere?
Multimea lor e notata FV(M) si FV(x) = {x}

FV(M N) = FV(M) reunit cu FV(N)
FV(λx.M) = FV(M) \ {x}

Ce inseamna sa redenumim o variabila intr-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.

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

Obs: inlocuim toate aparitiile lui x cu y, indiferent daca e libera, legata sau de legare si se foloseste doar daca y nu apare deja in M

α-echivalenta = cea mai mica relatie de congruenta pe multimea lambda termenilor

λx.M =$_\alpha$ λ.(M<y/x>)

Se satisfac urmatoarele reguli:

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

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

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

Substitutii = vrem sa substituim variabile cu lambda termeni

1. Vrem sa inlocuim doar variabilele libere
2. Nu vrem sa legam variabile libere neintentionat

Definim substitutia aparitiilor libere ale lui x cu N in M (M[N/x]):

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

Cursul 3

- **β-reductie** = procesul de a evalua lambda termeni prin pasarea de argumente functiilor
- **β-redex** = un termen de forma (λx.M)N
- redusul unui redex (λx.M)N = M[N/x]
- forma normala = un lambda termen fara redex-uri

Conventie: spunem ca doi termeni sunt egali daca sunt alfa-echivalenti.

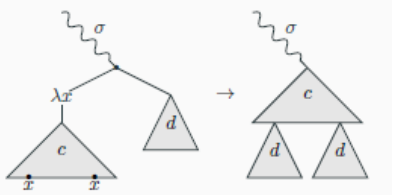
Reducem lambda termeni prin gasirea unui subtermen care este redex si apoi inlocuirea acelui redex cu redusul sau. Repetam acest proces de cate ori putem, pana nu mai sunt redex-uri.

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

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

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

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



Exemple + Observatii:

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

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

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

- reducerea unui redex poate crea noi redex-uri sau sterge alte redex-uri
- numarul de pasi necesari poate varia
- rezultatul final nu depinde de alegerea redex-urilor

$$
\begin{array}{lll}
\omega \equiv \underline{(\lambda x.x\,x)\,(\lambda y.y\,y)} & \to_\beta & (\lambda y.y\,y)\,(\lambda y.y\,y) \equiv \omega \\
& \to_\beta & \ldots
\end{array}
$$

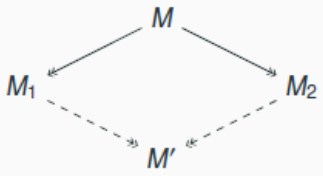
- exista lambda termeni care nu pot fi redusi
- lungimea unui termen nu trebuie sa scada in acest proces (creste sau ramane neschimbata)

$$
\begin{array}{lll}
\underline{(\lambda xy.y)\,((\lambda o.o\,o)\,(\lambda p.p\,p))}\,(\lambda z.z) & \to_\beta & \underline{(\lambda y.y)\,(\lambda z.z)} \\
& \to_\beta & \lambda z.z
\end{array}
$$

$$
\begin{array}{lll}
(\lambda xy.y)\,(\underline{(\lambda o.o\,o)\,(\lambda p.p\,p)})\,(\lambda z.z) & \to_\beta & (\lambda xy.y)\,(\underline{(\lambda p.p\,p)\,(\lambda p.p\,p)})\,(\lambda z.z) \\
& \to_\beta & \ldots
\end{array}
$$

- exista lambda termeni care desi pot fi redusi la o forma normala, pot sa nu o atinga niciodata

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 $M_2 \twoheadrightarrow_\beta M'$.



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

$M \twoheadrightarrow_\beta M'$ = M poate fi β-redus pana la M' in 0 sau mai multi pasi

- M slab normalizabil = exista N in forma normala astfel incat $M \twoheadrightarrow_\beta N$
- M puternic normalizabil = nu exista reduceri infinite care incep din M

Orice termen puternic normalizabil este si slab normalizabil

STRATEGII DE EVALUARE:

1. Strategia normala = alegem redex-ul cel mai din stanga care nu e continut de alt redex

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

2. Strategia aplicativa = alegem redex-ul cel mai din stanga care nu contine alte redex-uri

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

3. Strategia call-by-name (CBN) = strategia normala fara a face reduceri in corpul unei λ-abstractizari

4. Strategia call-by-value (CBV) = strategia aplicativa fara a face reduceri in corpul unei λ-abstractizari

CBV	CBN
Majoritatea limbajelor de programare functionala	Haskell
Functiile pot fi evaluate doar prin valori	Amanam evaluarea argumentelor cat mai mult (lazy evaluation)

Strategia CBV:

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

Strategia CBN:

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

Expresivitatea λ-calculului

- Valori booleene (Bool)
- Valori optiune (Maybe a)
- Perechi (Pair a b)
- Liste (List l)
- Numere naturale

Pentru Bool, definim T si F:

Bool ifTrue ifFalse b = b ifTre ifFalse

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

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

Pentru Maybe, definim Nothing si Just:

$\mathbf{Nothing} \triangleq \lambda nj.n$ (dintre cele două alternative o alege pe prima)

$\mathbf{Just} \triangleq \lambda anj.ja$ (**Just** *a* aplică al doilea argument valorii *a*)

Pentru Pair, definim:

$\mathbf{Pair} \triangleq \lambda abf.fab$ (**Pair** *a b* aplică funcția valorilor încapsulate)

Pentru List, definim:

$\mathbf{Nil} \triangleq \lambda fi.i$ (alege valoarea inițială)

$\mathbf{Cons} \triangleq \lambda alfi.fa(lfi)$ (**Cons** *a l* agregează lista, apoi agregheză valorea *a* în rezultat)

Pentru numere naturale:

$\mathbf{Zero} \triangleq \lambda fi.i \qquad \mathbf{Succ} \triangleq \lambda nfi.f(nfi) \qquad \mathbf{iterate} \triangleq \lambda fin.nfi$

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

Numeralul Church $\overline{n}$ este forma normală a λ-termenului $\mathbf{Succ}^n\, \mathbf{Zero}$, adică $\lambda fi.f^n\, i$, unde f^n reprezintă compunerea lui f cu ea însăși de n ori:

$$\begin{aligned} \overline{0} &\triangleq \lambda fi.f^0\, i = \lambda fi.i \\ \overline{1} &\triangleq \lambda fi.f^1\, i = \lambda fi.f\, i \\ \overline{2} &\triangleq \lambda fi.f^2\, i = \lambda fi.f\,(f\, i) \\ \overline{3} &\triangleq \lambda fx.f^3\, i = \lambda fi.f\,(f\,(f\, i)) \\ &\vdots \\ \overline{n} &\triangleq \lambda fi.f^n\, i = \lambda fi.\underbrace{f(f(\ldots(f}_{n}\, i)\ldots)) \end{aligned}$$

Obs: Succ pe argumentul n returneaza o functie care primeste ca argument o functie f, îi aplica n pentru a obtine compunerea de n ori a lui f cu ea însasi, apoi aplica iar f pentru a obtine compunerea de n + 1 ori a lui f cu ea insasi.

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

Acum, putem defini adunarea:

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

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

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

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

Putem defini adunarea și ca aplicarea repetată a funcției succesor:

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

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

Pentru a defini inmultirea:

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

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

In plus, putem avea o functie de la numere naturale la booleeni care verifica daca un numar natural este 0 sau nu:

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

Cursul 4

Puncte fixe = x punct fix al lui f daca f(x) = x
- o functie poate avea mai multe puncte fixe sau chiar o infinitate (f(x)=x)

Am notat cu M $\twoheadrightarrow_\beta$ M' faptul ca M poate fi β-redus pâna la M' în 0 sau mai multi pasi de β-reductie.
$\twoheadrightarrow_\beta$ este închiderea reflexiva si tranzitiva a relatiei $\rightarrow_\beta$

Notam cu M $=_\beta$ M' faptul ca M poate fi transformat în M' în 0 sau mai multi pasi de β-reductie, transformare în care pasii de reductie pot fi si întorsi.
$=_\beta$ este închiderea reflexiva, simetrica si tranzitiva a relatiei $\rightarrow_\beta$.

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

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

Daca F si M sunt λ-termeni, spunem ca M este un punct fix al lui F daca F M $=_\beta$ M.
Teorema: În lambda calculul fara tipuri, orice termen are un punct fix.

Combinatorii de puncte fixe = termeni inchisi care construiesc un punct fix pentru un termen arbitrar

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
$\Theta F \twoheadrightarrow_\beta F\,(\Theta\,F)$.

Punctele fixe ne permit sa rezolvam ecuatii.
Un model de ecuatie:
fact n = if (isZero n) (1) (mul n (fact(pred n)))

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.

Cursul 5

De ce nu e prea ok lambda calcul fara tipuri?

- Aplicari xx sau MM sunt permise, dar sunt contraintuitive
- Existenta formelor normale pentru λ-termeni nu este garantata si putem avea calcule infinite
- Orice λ-termen are un un punct fix -> nu e in armonie cu ceea ce stim despre functii oarecare

Avem o multime de tipuri variabila V = {α, β, γ, ...}.
Multimea tipurilor simple T e definita prin T = V | T -> T

- Tipul variabila: daca α ∈ V, atunci ∈ T
- Tipul sageata: daca σ, τ ∈ T, atunci (σ -> τ) ∈ T

Tipurile **variabila** = reprezentari abstracte pentru tipuri de baza cum ar fi Nat sau List
Tipurile **sageata** = reprezentari pentru tipuri de functii precum (Nat -> Real)

IN TIPUL SAGEATA, PARANTEZELE SUNT ASOCIATIVE LA DREAPTA.

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 $\lambda x.\, M : \sigma \to \tau$.

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.

Asociativitatea la dreapta pentru tipurile sageata VS Asociativitatea la stanga pentru aplicare

Asociere explicita (Church-typing)	Asociere implicita (Curry-typing)
• Consta în prescrierea unui unic tip pentru fiecare variabila, la introducerea acesteia. • Presupune ca tipurile variabilelor sunt explicit stabilite. • Tipurile termenilor mai complecsi se obtin natural, tinând cont de conventiile pentru aplicare si abstractizare.	• Consta în a nu prescriere un tip pentru fiecare variabila, ci în a le lasa "deschise" (implicite). • În acest caz, termenii typeable sunt descoperiti printr-un proces de cautare, care poate presupune "ghicirea" anumitor tipuri.

Multimea λ-termenilor cu pre-tipuri Λ_T este

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

Definitii:

- **Afirmatie** = expresie de forma M:σ, unde M ∈ Λ_T si σ ∈ T (M = subiect si σ = tip)
- **Declaratie** = o afirmatie in care subiectul e o variabila (x :σ)
- **Context** = lista de declaratii cu subiecti diferiti
- **Judecata** = o expresie Γ ⊢ M:σ, unde Γ este context si M:σ este o afirmatie

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

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

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

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

Un exemplu de exercitiu:
Sa aratam ca termenul λy : α → β. λz : α. yz are tipul (α → β) → α → β în contextul vid.

Putem reprezenta in mai multe moduri rezolvarea:

1. Stilul arbore

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

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

2. Stilul liniar

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

3. Stilul cutii

1.	$y:\alpha \to \beta$	(*context*)
2.	$z:\alpha$	(*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

Cursul 6

Ce **probleme** putem sa rezolvam in teoria tipurilor?

1. **Type Checking** = putem gasi o derivare pentru un context si un tip
   context ⊢ term: type
2. **Well-typedness** (Typability) = verificam daca un termen este legal (cautam un context si un tip)
   ? ⊢ term: ?
3. **Type Assignment** = Typability, dar contextul este dat si trebuie sa gasim tipul
   context ⊢ term: ?
4. **Term Finding** = avem contextul si tipul, trebuie sa stabilim daca exista un termen cu acel tip dat
   context ⊢ ? : type

Toate aceste probleme sunt decidabile pentru calculul Church λ→.

Ce **limitari** avem?

1. Nu mai avem recursie nelimitata (combinatorii de punct fix nu sunt typeable).
   Y ≜ λy. (λx. y (x x)) (λx. y (x x)) nu este typeable.
   Dar avem recursie primitiva (permite doar looping in care nr de iteratii este cunoscut dinainte):
   add ≜ λmnfx.mf (n f x)
2. Tipurile pot fi prea restrictive (ca solutii avem let-polymorphism + cuantificatori de tipuri)
   (λf . if (f T) (f 3) (f 5)) (λx. x) ar trebui sa aiba un tip, dar nu are!

Multimea tipurilor:
T = V | T → T | **Unit**

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

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

Multimea tipurilor:

T = V | T → T | Unit | **Void**

Nu exista regula de tipuri pentru deoarece tipul Void nu are inhabitant (e la fel ca mai sus).

Multimea tipurilor:

T = V | T → T | Unit | Void | **TxT**

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

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

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

Multimea tipurilor:

T = V | T → T | Unit | Void | TxT | **T + T**

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

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

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

Corespondenta Curry-Howard

Teoria Tipurilor	Logica
Tipuri	Formule
Termeni	Demonstratii
Inhabitation a tipului σ	Demonstratie a lui σ
Tip produs	Conjunctie
Tip functie	Implicatie
Tip suma	Disjunctie
Tipul void	False
Tipul unit	True

Corectitudine = sintaxa implica semantica

Completitudine = sintaxa si semantica coincid

λ-calcul cu tipuri Deductie naturala

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

Faptul ca exista un termen de tip σ (inhabitation of type σ) inseamna ca σ este teorema sau ca are o demonstratie in logica.

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

Propositions are types! ♡

λ-calcul cu tipuri	Deducție naturală
$\dfrac{\{x : \sigma\} \vdash x : \sigma}{\vdash \lambda x.\, x : \sigma \to \sigma}\ (\to_I)$	$\dfrac{\{\sigma\} \vdash \sigma}{\vdash \sigma \supset \sigma}\ (\supset_I)$
$\dfrac{\dfrac{\overline{\{x : \sigma, y : \tau\} \vdash x : \sigma}}{\{x : \sigma\} \vdash \lambda y.\, x : \tau \to \sigma}\ (\to_I)}{\vdash \lambda x.\, (\lambda y.\, x) : \sigma \to (\tau \to \sigma)}\ (\to_I)$	$\dfrac{\dfrac{\overline{\{\sigma, \tau\} \vdash \sigma}}{\{\sigma\} \vdash \tau \to \sigma}\ (\supset_I)}{\vdash \sigma \to (\tau \to \sigma)}\ (\supset_I)$

Proofs are Terms! ♡

Demonstrațiile sunt termeni!

Formulele de mai jos nu sunt demonstrabile in logica intuitionista:

- dubla negatie: ¬¬ϕ ⊃ ϕ
- excluded middle: ϕ ∨ ¬ϕ
- legea lui Pierce: ((ϕ ⊃ τ) ⊃ ϕ) ⊃ ϕ

Nu exista semantica cu tabele de adevar pentru logica intuitionista!

Initial, corespondenta Curry-Howard a fost intre calculul Church λ → si sistemul de deductie naturala al lui Gentzen pentru logica intuitionista.

Fundamentele limbajelor de programare

Notițe pentru parțial

Aprilie, 2023

1 Paradigme de definire a funcţiilor

Criteriul/Paradigma	Abordarea extensională	Abordarea intensională
Definirea unei funcții	$f : X \to Y$ este o mulțime de perechi f $\subseteq X \times Y$, a.î., $\forall x \in X, \exists y \in Y$ a.î.$(x, y) \in f$	O formula de calcul pentru funcție de ex., $f(x) = x^2 - 1$
Eglitatea funcțională	$f = g \iff f(x) = g(x), \forall x \in X$ Se compară doar rezultatele funcției	$f(x) = x^2 - 1, g(x) = (x-1)(x+1), f \neq g$ Se compară formulele de definiție
Puncte forte	Cuprinde și funcții care nu pot fi definite prin formule	Utilă d.p.d.v al implementării.

2 Introducere în $\lambda - calcul$

Exemple de reprezentare ale funcțiilor în $\lambda - calcul$:

Reprezentarea clasică	λ calcul
$f(x) = x^2$	$\lambda x.x^2$
$g(x) = (f \circ f)(x)$	$\lambda x.f(f(x))$
$h(f) = f \circ f$ (funcție de nivel înalt)	$\lambda f.\lambda x.f(f(x))$

3 Noțiuni preliminare despre tipuri

Definitia 3.1. *Notăm cu x:X, faptul că un termen x are tipul X.*

Exemplul 3.2. $f = \lambda x.x : X \to X$, $g = \lambda f.\lambda x.f(f(x)) : (X \to X) \to (X \to X)$.

Comparație între versiunile de $\lambda - calcul$, in funcție de prezența tipurilor:

Fără tipuri	Tipuri simple	Tipuri polimorfice
Nu dăm tipul expresiilor	Dăm tipul expresiilor	Putem specifica tipul $X \to X$, fără a-l preciza explicit pe X
Nu dăm domeniul/codomeniul funcțiilor	Nu putem aplica o funcție, decât dacă se potrivește tipul argumentului	

4 Calculabilitate

Definitia 4.1. *Spunem că o funcție este calculabilă, dacă există o metodă de a calcula f(n), pentru orice n.*

Modele de calculabilitate, echivalente:

(i) Mașina Turing (Turing)

(ii) Funcții recursive (Goedel)

(iii) $\lambda - calcul$ (Church)

5 Noțiuni de $\lambda - calcul$

Definitia 5.1. *Expresiile din $\lambda - calcul$ se numesc termeni.*

Notatia 5.2. *Notăm cu $\mathbb{V}$, mulțimea infinită a variabilelor si cu A alfabetul, $A = \mathbb{V} \cup \{"(", ")", "\lambda", "."\}$.*

Definitia 5.3 (Definiția inductivă a λ-termenilor, scrisă în forma BNF)**.**

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

Definitia 5.4 (Definiție alternativă a λ-termenilor)**.**
Mulțimea λ-termenilor este cea mai mică submulțime $\Lambda \subseteq A^*$, a.î.:

(i) $\mathbb{V} \subseteq \Lambda$

(ii) *Dacă* $M, N \in \Lambda$, *atunci* $MN \in \Lambda$.

(iii) *Dacă* $x \in \mathbb{V}, M \in \Lambda$, *atunci* $(\lambda x.M) \in \Lambda$.

Conventia 5.5.

(i) *Parantezele exterioare se elimină.*

(ii) *Aplicația este asociativă la stânga.*

(iii) *Corpul abstractizătilor se extinde la dreapta, cât de mult posibil.*

(iv) *Abstractizările se comprimă.*

6 Variabile libere și variabile legate

Definitia 6.1. *Terminlogie specifică:*

(i) $\lambda\_.\_$ *se numește operator de legare (binder).*

(ii) x *din* $\lambda x.\_$ *se numește variabilă de legare (binding).*

(iii) N *din* $\lambda x.N$ *se nume'te domeniul de legare al lui x(scope). Toate aparițiile lui* x *în* N *sunt legate.*

(iv) *O variabilă care nu este legată se numește liberă.*

(v) *Un termen care nu conține variabile libere se numește termen închis, sau combinator.*

Notatia 6.2. *Mulțimea variabilelor libere ale unui termen se notează cu FV.*

Definitia 6.3 (Definiția recursivă pe termeni a mulțimii variabilelor libere)**.**

(i) $FV(x) = x$

(ii) $FV(MN) = FV(M) \cup FV(N)$

(iii) $FV(\lambda x.M) = FV(M) \setminus x$

7 Redenumire de variabile

Notatia 7.1. *Fie* $x, y \in \mathbb{V}$ *și* M *un termen. Atunci, notăm cu* $M\langle y/x\rangle$ *termenul obținut prin redenumirea lui* x *cu* y *în* M.

(i) $x\langle y/x\rangle \equiv y$

(ii) $z\langle y/x\rangle \equiv z$, dacă $x \neq z$

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

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

(v) $(\lambda z.M)\langle y/x\rangle \equiv \lambda z.(M\langle y/x\rangle)$, dacă $x \neq z$

Acest tip de redenumire se folosește doar în cazurile în care y nu apare în M.

8 α-echivalență

Definitia 8.1. *α-echivalența este cea mai mică relație de congruență pe λ-termeni, care satisface următoarele:*

$$(\text{REFL}) \quad \frac{}{M = M}$$

$$(\text{SYMM}) \quad \frac{M = N}{N = M}$$

$$(\text{TRANS}) \quad \frac{M = N \quad N = P}{M = P}$$

$$(\text{CONG}) \quad \frac{M = P \quad N = Q}{MN = PQ}$$

$$(\xi) \quad \frac{M = N}{\lambda x.M = \lambda x.N}$$

$$(\alpha) \quad \frac{y \notin M}{\lambda x.M = \lambda y.(M\langle y/x\rangle)}$$

Definitia 8.2 (Definiție alternativă)**.**
Numim α-echivalență egalitatea între termeni, modulo redenumiri de variabile legate.

Conventia 8.3. *Convenția Barendregt: Variabilele legate sunt redenumite, pentru a fi distincte.*

9 Substituții

Notatia 9.1. *Fie M,N termeni, $x \in \mathbb{V}$. Notăm cu $M[N/x]$, rezultatul obținut prin înlocuirea lui x cu N în M.*

Reguli pentru substituție:

(i) Inlocuim doar variabilele libere.

(ii) Pentru a evita legarea neintenționată a variabilelor libere, redenumim variabilele legate, înainte de substituție.

(i) $x[N/x] \equiv N$

(ii) $y[N/x] \equiv y$

(iii) $MP[N/x] \equiv (M[N/x])(P[N/x])$

(iv) $(\lambda x.M)[N/x] \equiv \lambda x.M$

(v) $(\lambda y.M)[N/x] \equiv \lambda y.(M[N/x])$, dacă $x \neq y$ și $y \notin FV(N)$.

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

10 β-reducție

Conventia 10.1. *Doi termeni sunt egali, dacă sunt α-echivalenți.*

Definitia 10.2. *Terminlogie specifică:*

(i) Se numește β-reducție, procesul de evaluare a λ-termenilor, prin "pasarea de argumente funcțiilor".

(ii) Se numește β-redex, un termen de forma $(\lambda x.M)N$.

(iii) Redusul unui termen $(\lambda x.M)N$ este $M[N/x]$.

(iv) Se numește formă normală un λ-termen fără redex-uri.

Definitia 10.3. *Un pas de β-reducție este cea mai mică relație pe λ-termeni, care satisface următoarele:*

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

$$(\text{CONG1}) \quad \frac{M \to_\beta P}{MN \to_\beta PN}$$

$$(\text{CONG2}) \quad \frac{N \to_\beta P}{MN \to_\beta MP}$$

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

Procesul de β-reducție constă în aplicarea repetată a câte unui pas de β-reducție, rezultatul final nefiind afectat de ordinea alegerii redex-urilor.

Notatia 10.4. *Notăm cu $M \twoheadrightarrow M'$, faptul că, prin efectuarea a 0 sau mai mulți pași de β-reducție asupra lui M, se poate obține M'.*

Definitia 10.5. *Un termen M se numește:*

(i) Slab-normalizabil, dacă $\exists N$ formă normală, a.î. $M \twoheadrightarrow N$.

(ii) Puternic-normalizabil, dacă nu există β-reducții infinite care încep din M.

Teorema 10.6 (Confluența β-reducției)**.** ***Teorema Church-Rosser***
Dacă $M \twoheadrightarrow M_1$ și $M \twoheadrightarrow M_2$, atunci $\exists M'$, a.î. $M_1 \twoheadrightarrow M'$ și $M_2 \twoheadrightarrow M'$.

Consecinta 10.7. *Un λ-termen are cel mult o β-formă normală (modulo α-echivalență).*

10.1 Strategii de evaluare

(i) **Strategia normală (leftmost-outermost)**: se alege, la fiecare pas, redex-ul cel mai din stânga și apoi cel mai din exterior. Avantajul acestei strategii constă în faptul că, dacă există o formă normală pentru termenul evaluat, strategia garantează găsirea acestuia.

(ii) **Strategia aplicativă (leftmost-innermost)**: se alege, la fiecare pas, redex-ul cel mai din stânga și apoi cel mai din interior.

10.1.1 Strategii în programarea funcţională

(i) **Strategia CBN**: folosește strategia normală de β-reducție, fără reduceri în corpul λ-abstractizărilor. Amânăm evaluarea argumentelor (lazy-evaluation).

(ii) **Strategia CBV**: folosește strategia aplicativă de β-reducție, fără reduceri în corpul λ-abstractizărilor. Funcțiile sunt apelate doar prin valoare (după ce argumentele au fost complet evaluate).

Haskell este singurul limbaj de programare funcțional care implementeaza Strategia CBN.

Definitia 10.8. *In acest context, numim valoare orice λ-termen pentru care nu există β-reducții.*

Exemplul 10.9. $(\lambda x.x)$ este o valoare, în timp ce $(\lambda x.x)1$ nu este.

11 Expresivitatea λ-calculului

11.1 Reprezentarea valorilor booleene

$$T = \lambda xy.x \qquad F = \lambda xy.y \qquad if = \lambda btf.btf$$
$$and = \lambda b_1 b_2.if\, b_1 b_2 F \qquad or = \lambda b_1 b_2.if\, b_1 T b_2 \qquad not = \lambda b_1.if\, b_1 F T$$

11.2 Reprezentarea numerelor naturale, sub forma numeralilor Church

$$\bar{n} = \lambda fx.f^n x$$
$$Succ = \lambda nfx.f(nfx) \qquad add = \lambda mnfx.mf(nfx) \qquad mul = \lambda mn.m(add\ n)\bar{0} \qquad exp = \lambda mn.m(mul\ n)\bar{1}$$
$$isZero = \lambda nTF.n(\lambda z.F)T$$

12 Puncte fixe

Definitia 12.1 (Definiția generală a punctului fix)**.**
*Fie f o funcție. Spunem că x este un **punct fix** al lui f, dacă $f(x) = x$.*

Notatia 12.2. *Fie M și N termeni. Spunem că $M =_\beta M'$, dacă N poate fi obținut în 0 sau mai mulți pași direcți sau inverși de β-reducție din M.*

Exemplul 12.3. $(\lambda y.yv)z =_\beta (\lambda x.zx)v$, deoarece avem:

$$(\lambda y.yv)z \to_\beta zv_\beta \leftarrow (\lambda x.zx)v,$$
unde cu $_\beta \leftarrow$ am notat inversa relației de β-reducție.

Definitia 12.4 (Definiția punctului fix în λ-calcul)**.**
*Dacă F și M sunt λ-termeni, spunem că M este un **punct fix** al lui F, dacă $FM =_\beta M$.*

Teorema 12.5. *In λ-calcul fără tipuri, orice termen are un punct fix, și anume $M = (\lambda x.F(xx))(\lambda x.F(xx))$.*

Definitia 12.6. ***Combinatorii de puncte fixe*** *sunt termeni închiși care construiesc un punct fix pentru un termen arbitrar.*

Exemplul 12.7. Combinatorul de punct fix al lui **Curry**: $\mathbf{Y} = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$
Pentru orice termen F, $\mathbf{Y}$ este un punct fix al lui F, deoarece $\mathbf{Y}F \twoheadrightarrow F(\mathbf{Y}F)$.

Exemplul 12.8. Combinatorul de punct fix al lui **Turing**: $\mathbf{\Theta} = (\lambda xy.y(xxy))(\lambda xy.y(xxy))$
Pentru orice termen F, $\mathbf{\Theta}$ este un punct fix al lui F, deoarece $\mathbf{\Theta}F \twoheadrightarrow F(\mathbf{\Theta}F)$.

12.1 Rezolvarea de ecuații în λ-calcul

Putem rezolva ecuații, cu ajutorul punctelor fixe (conform Teoremei 12.5, în λ-calcul fără tipuri, orice termen are un punct fix).

Exemplul 12.9. Considerăm funcția factorial, definită astfel: $fact\ n = if(isZero\ n)(\bar{1})(mul\ n(fact(pred\ n)))$.
Rescriem ecuația, astfel: $fact = (\lambda fn.if(isZero\ n)(\bar{1})(mul\ n(f(pred\ n))))fact$.
Notăm termenul $\lambda fn.if(isZero\ n)(\bar{1})(mul\ n(f(pred\ n)))$ cu F și ecuația devine: $fact = F\ fact$ - o ecuație de punct fix, pe care o putem rezolva, luând:

$$fact = \mathbf{Y}F = \mathbf{Y}(\lambda fn.if(isZero\ n)(\bar{1})(mul\ n(f(pred\ n))))$$

13 λ-calcul cu tipuri simple

Observatia 13.1. *Limitări ale λ-calculului fără tipuri:*

(i) Sunt permise aplicări de forma MM, care sunt contraintuitive.

(ii) Nu este garantată exsitenâ formelor normale pentru λ-termeni, ceea ce poate conduce la calcule infinite.

(iii) Orice λ-termen are un punct fix, ceea ce nu este valabil pentru funcții oarecare.

Notatia 13.2. *Notăm cu $\mathbb{V}$ mulțimea infinită a **tipurilor variabilă**.*
De obicei, vom folosi simbolurile $\alpha, \beta, \gamma \ldots$, pentru tipuri variabilă, dar, uneori, folosim și A, B, ...

Definitia 13.3. *(i) Mulțimea **tipurilor simple** este definită prin:*

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

*(a) **Tipuri variabilă**: Dacă $\alpha \in \mathbb{V}$, atunci $\alpha \in \mathbb{T}$. (reprezintă tipuri de bază)*

*(b) **Tipuri săgeată**: Dacă $\sigma, \tau \in \mathbb{T}$, atunci $\sigma \to \tau \in \mathbb{T}$ (reprezintă tipuri pentru funcții).*

Observatia 13.4. *Parantezele din tipurile săgeată sunt asociative la dreapta.*

Notatia 13.5. *Not m cu $M : \sigma$, faptul că termenul M are tipul σ.*

Observatia 13.6. *Orice variabilă are un tip unic.*

13.1 Termeni și tipuri

(i) (**Variabilă**) $x : \sigma$

(ii) (**Aplicare**) Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $MN : \tau$.

(iii) (**Abstractizare**) Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.M : \sigma \to \tau$.

Definitia 13.7. *M are tip (este typeable), dacă există un tip σ, a.î. $M : \sigma$.*

Exemplul 13.8. Exemplu de termen care nu are tip: xx (deoarece x ar trebui să aibă concomitent tipurile $\sigma \to \tau$ și τ, ceea ce este imposibil).

13.1.1 Church-typing vs. Curry-typing

Church-typing (asociere implicită)	**Curry-typing (asociere explicită)**
Dăm tipurile variabilelor, la introducere	Nu prescriem tipurile variabilelor, trebuie deduse

Observatia 13.9. *Asocierea implicită are proprietatea de a găsi cele mai generale tipuri posibile.*
Totuși, în continuare, vom folosi Church-typing, marcând tipurile ***variabilelor legate****, după introducerea lor cu o abstractizare, și menționând tipurile* ***variabilelor libere*** *într-un* ***context****.*

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

Definitia 13.10. *Mulțimea λ-termenilor cu pre-tipuri este:*

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

Definitia 13.11. *Terminologie:*

(i) *O* ***afirmație*** *este o expresie de forma $M : \sigma$, unde $M \in \Lambda_{\mathbb{T}}$ și $\sigma \in \mathbb{T}$. M se numește* ***subiect*** *și σ* ***tip****.*

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

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

(iv) *O* ***judecată*** *este o expresie de forma $\Gamma \vdash M : \sigma$, unde Γ este un context și $M : \sigma$ este o afirmație.*

(v) *Un termen M în calculul Church $\lambda \rightarrow$ este* ***legal****, dacă există un context Γ și un tip σ, a.î. $\Gamma \vdash M : \sigma$.*

Sistemul de deducție:

$$(\text{VAR}) \quad \frac{}{\Gamma \vdash x : \sigma} \text{ if } x : \sigma \in \Gamma$$

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

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

13.3 Probleme din teoria tipurilor

Următoarele probleme sunt decidabile, pentru calculul Church $\lambda \rightarrow$:

(i) Type checking = verificarea faptului că, date fiind un context, un termen și un tip dat, termenul poate avea tipul respectiv în acel context: $context \vdash term : type$

(ii) Typability = veriricarea legalității: $? \vdash term :?$

(iii) Type assignment = typability în care se dă și contextul: $context \vdash term :?$

(iv) Term finding (inhabitation) = dându-se un context și un tip, trebuie stabilit dacă există un termen cu acel tip, în contextul dat: $context \vdash ? : type$

13.4 Limitări ale λ-calculului cu tipuri simple

(i) Nu avem recursie nelimitată (deoarece combinatorii de punct fix nu sunt typeable). Avem doar **recursie primitvă** (cu număr cunoscut de pași).

(ii) Tipurile pot fi prea restrictive $\rightarrow$ soluția adoptată: **cuantificatori de tipuri** $(\lambda x.x : \Pi\alpha.\alpha \rightarrow \alpha)$

13.5 Alte tipuri

Adăugăm tipurile Unit, Void $\times$, $+$, la mulțimea de tipuri , termenii construiți cu ele, la mulțimea λ-termenilor cu pre-tipuri, și, pentru fiecare tip nou adăugat, introducem regulile de inferență corespunzătoare:

$$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T} \mid Unit \mid Void \mid \mathbb{T} \times \mathbb{T} \mid \mathbb{T} + \mathbb{T}$$
$$\Lambda_{\mathbb{T}} = x \mid \Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}} \mid \lambda x : \mathbb{T}.\Lambda_{\mathbb{T}} \mid unit \mid \langle \Lambda_{\mathbb{T}}, \Lambda_{\mathbb{T}} \rangle \mid fst\Lambda_{\mathbb{T}} \mid snd\Lambda_{\mathbb{T}} \mid Left\Lambda_{\mathbb{T}} \mid \Lambda_{\mathbb{T}} \mid case\ \Lambda_{\mathbb{T}} of \Lambda_{\mathbb{T}}; \Lambda_{\mathbb{T}}$$

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

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

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

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

$$(+_{I_1}) \quad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash Left\ M : \sigma + \tau}$$

$$(+_{I_2}) \quad \frac{\Gamma \vdash M : \tau}{\Gamma \vdash Right\ M : \sigma + \tau}$$

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

13.6 Corespondența Curry-Howard

Definitia 13.12. *O formulă este **tautologie**, dacă are valoarea 1, pentru toate evaluările posibile.*

Definitia 13.13.

(i) Corectitudine = sintaxa implica semantica

(ii) Completitudine = sintaxa coincide cu semantica

λ-calcul cu tipuri	Deducție naturală
$(\times_I) \quad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma, \tau}$	$(\wedge_I) \quad \frac{\Gamma \vdash \sigma \quad \Gamma \vdash \tau}{\Gamma \vdash \wedge\sigma\tau}$
$(\times_{E_1}) \quad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash fstM : \sigma}$	$(\wedge_{E_1}) \quad \frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \sigma}$
$(\times_{E_2}) \quad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash sndM : \tau}$	$(\wedge_{E_2}) \quad \frac{\Gamma \vdash \sigma \wedge \tau}{\Gamma \vdash \tau}$
$(\to_I) \quad \frac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}$	$(\supseteq_I) \quad \frac{\Gamma \cup \{\sigma\} \vdash \tau}{\Gamma \vdash \sigma \to \tau}$
$(\to_E) \quad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}$	$(\supseteq_E) \quad \frac{\Gamma\sigma \supseteq \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau}$

Teoria tipurilor	Logică
tipuri	formule
termeni	demonstrații
Inhabitation pentru tipul σ	demonstrație pentru σ
tip produs	conjuncție
tip sumă	disjuncție
tip funcție	implicație
tip Void	fals
tip Unit	adevăr