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:

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

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 $b = false$.

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

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:

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

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

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

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 $\lambda \to$:

- $\Gamma \vdash x : \sigma$ (var), dacă $x : \sigma$
- Din $\Gamma \vdash M : \sigma \to \tau$ și $\Gamma \vdash N : \sigma$ rezultă $\Gamma \vdash MN : \tau$. $(\to_i)$
- Din $\Gamma, x : \sigma \vdash M : \tau$ rezultă $\Gamma \vdash (\lambda x : \sigma.M) : \sigma \to \tau$. $(\to_e)$

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

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

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

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

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

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

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

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

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

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

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)