# Lambda calcul

Lambda calculul (engleză: lambda calculus) este un model de calcul care surprinde esența programării funcționale. Lambda-calculul este un limbaj de programare aparent foarte simplu (cu doar trei construcții sintactice), dar care este complet, în sensul în care poate descrie orice calcul efectuat de un calculator.

În lambda calcul, lucrăm cu o mulțime infinit numărabilă de identificatori (sau variabile), care sunt notate de obicei în felul următor:

$$Id = \{\mathsf{x}, \mathsf{y}, \mathsf{x}', \mathsf{y}', \mathsf{z}_1, \ldots\}.$$

Programele din lambda-calcul se numesc *lambda-termeni* (sau *lambda-expresii*).

## 1 Sintaxa

Mulțimea *lambda-termenilor* este cea mai mică mulțime care are următoarele proprietăți:

1. orice identificator este un lambda-termen;
2. dacă $x$ este un identificator și $t$ este un lambda-termen, atunci $(\lambda x.t)$ este un lambda-termen;
3. dacă $t_1$ și $t_2$ sunt lambda-termeni, atunci $(t_1\ t_2)$ este un lambda-termen.

Termenii construiți folosind a doua regulă se numesc *lambda-abstracții*, iar termenii care sunt construiți cu ultima regulă *lambda-aplicații*.

Exemple de lambda-termeni:

$\mathsf{x}$, $\mathsf{y}$, $\mathsf{z}$, $(\mathsf{x}\ \mathsf{x})$, $(\lambda\mathsf{x}.\mathsf{x})$, $(\lambda\mathsf{x}.(\lambda\mathsf{y}.\mathsf{x}))$, $(\lambda\mathsf{x}.(\lambda\mathsf{y}.(\mathsf{x}\ \mathsf{x})))$, $(\lambda\mathsf{x}.((\lambda\mathsf{y}.\mathsf{x})\ \mathsf{x}))$,

$((\lambda\mathsf{x}.\mathsf{x})\ (\lambda\mathsf{x}.\mathsf{x}))$, $((\lambda\mathsf{x}.\mathsf{y})\ (\lambda\mathsf{z}.(\mathsf{x}\ \mathsf{x})))$, $((\lambda\mathsf{x}.(\lambda\mathsf{y}.\mathsf{x}))\ ((\mathsf{x}\ \mathsf{y})\ \mathsf{z}))$

Exercițiu: dați exemplu de 5 lambda-termeni interesanți.

## 2 Notații

1. O secvență de lambda-abstracții se parantetizează implicit spre dreapta:

   Exemplu: $\lambda\mathsf{x}.\lambda\mathsf{y}.\lambda\mathsf{z}.\mathsf{x} = (\lambda\mathsf{x}.(\lambda\mathsf{y}.(\lambda\mathsf{z}.\mathsf{x})))$;

2. O secvență de lambda-aplicații se parantetizează implicit spre stânga:

   Exemplu: $\mathsf{x}\ \mathsf{y}\ \mathsf{z} = ((\mathsf{x}\ \mathsf{y})\ \mathsf{z})$;

3. O aplicație se extinde cât mai spre dreapta:

$$\text{Exemplu: } \lambda x.\lambda y.x\ y\ z = (\lambda x.(\lambda y.((x\ y)\ z))),$$

Atenție! Ultima regulă face ca în termenul $(\lambda x.\lambda y.x\ y)\ z$ parantezele să fie obligatorii (altfel aș înțelege alt termen).

Exercițiu: scrieți termenii din primul exemplu cu cât mai puține paranteze.

# 3 Variabile legate și variabile libere

Un lambda-termen poate avea 0 sau mai multe variabile legate și 0 sau mai multe variabile libere.

Variabilele legate sunt cele care apar imediat după semnul $\lambda$.

Funcția *bound*, definită ca mai jos, calculează mulțimea variabilelor legate ale unui lambda-termen:

1. $bound(x) = \emptyset$ (pentru orice identificator $x \in Id$);
2. $bound(\lambda x.t) = \{x\} \cup bound(t)$ (pentru orice identificator $x$ și orice lambda-termen $t$);
3. $bound(t_1\ t_2) = bound(t_1) \cup bound(t_2)$ (pentru orice lambda-termeni $t_1, t_2$).

De exemplu, $bound(\lambda x.\lambda y.\lambda x.x\ y\ z) = \{x, y\}$.

Funcția *free*, definită ca mai jos, calculează mulțimea variabilelor libere ale unui lambda-termen:

1. $free(x) = \{x\}$ (pentru orice identificator $x \in Id$);
2. $free(\lambda x.t) = free(t) \setminus \{x\}$ (pentru orice identificator $x$ și orice termen $t$);
3. $free(t_1\ t_2) = free(t_1) \cup free(t_2)$ (pentru orice termeni $t_1, t_2$).

De exemplu, $free(\lambda x.\lambda y.\lambda x.x\ y\ z) = \{z\}$.

Atenție! Există termeni unde același identificator este și variabilă liberă și variabilă legată. De exemplu, $(x\ (\lambda x.x))$.

Exercițiu: calculați variabilele libere și variabilele legate ale termenilor dați ca exemplu mai devreme.

# 4 Alfa-echivalență

Doi termeni sunt *alfa-echivalenți* (sau *$\alpha$-echivalenți*) dacă, intuitiv, unul poate fi obținut din celălalt prin redenumirea variabilelor legate, fără a îi schimba înțelesul.

De exemplu, $\lambda x.\lambda y.x$ este alfa-echivalent cu $\lambda x'.\lambda y'.x'$ și cu $\lambda y.\lambda x.y$.

Atenție! Termenul $\lambda x.\lambda y.x$ nu este alfa-echivalent cu $\lambda x.\lambda x.x$.

Faptul că doi termeni $t_1, t_2$ sunt alfa-echivalenți se notează cu $t_1 =_\alpha t_2$.

Exercițiu. Stabiliți care dintre următorii termeni sunt alfa-echivalenți:

1. $\lambda x.x =_\alpha \lambda y.y$?
2. $\lambda x.y =_\alpha \lambda y.x$?
3. $\lambda x.x\ y =_\alpha \lambda x.x\ z$?

4. $\lambda x.x\ y =_\alpha \lambda y.y\ y$?
5. $x\ \lambda x.x\ y =_\alpha x\ \lambda z.z\ y$?
6. $x\ \lambda x.x\ y =_\alpha y\ \lambda x.x\ y$?

# 5 Substituții

Cu $t[x/t']$ se notează *termenul obținut din $t$ prin substitutirea aparițiilor libere ale variabilei $x$ cu $t'$*.

De exemplu, $\lambda x.\lambda y.x\ y\ z[z/\lambda y'.y'] = \lambda x.\lambda y.x\ y\ (\lambda y'.y')$.

Alt exemplu: $x\ (\lambda x.x)[x/\lambda y'.y'] = (\lambda y'.y')(\lambda x.x)$.

Ultimul exemplu arată că se înlocuiesc doar aparițiile libere ale variabilelor.

Definiția substituției este:

1. $x[x/t'] = t'$;
2. $y[x/t'] = y$ (dacă $x \neq y$);
3. $(t_1\ t_2)[x/t'] = (t_1[x/t'])\ (t_2[x/t'])$;
4. $(\lambda x.t)[x/t'] = \lambda x.t$;
5. $(\lambda y.t)[x/t'] = \lambda y.(t[x/t'])$ (dacă $x \neq y$).

Observați cum în definiția din cazul al patrulea, ne asigurăm că nu înlocuim aparițiile legate ale variabilelor.

Substituția de mai sus se numește *substituție care capturează* (capturing substitution), deoarece are următorul comportament nedezirabil:

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

Observați că am pornit cu o funcție care își ignoră argumentul și întoarce variabila liberă $y$. Înlocuind $y$ cu $x$, obținem funcția identitate (care nu își mai ignoră argumentul). În acest caz, spunem că $\lambda x$ *capturează* variabila $x$ din codomeniul substituției.

# 6 Substituția care evită capturarea

Am văzut mai devreme că $(\lambda x.y)[y/x] = \lambda x.x$.

Dacă pornim cu un termen alfa-echivalent cu $(\lambda x.y)$, de exemplu cu $\lambda x'.y$, obținem un alt comportament al substituției:

$(\lambda x'.y)[y/x] = \lambda x'.x$ (nu mai obținem funcția identitate, ci doar funcția care își ignoră în continuare argumentul și întoarce $x$ în loc de $y$).

De această dată, rezultatul pare mai rezonabil. Substituția în care redenumim variabilele legate pentru a nu captura variabile libere din codomeniul substituției se numește *substituție care evită capturarea*.

Vom nota substituția care evită capturarea (capture-avoiding substitution) cu $t[\![x/t']\!]$. Definiția ei este:

1. $x[\![x/t']\!] = t'$;
2. $y[\![x/t']\!] = y$ (dacă $x \neq y$);
3. $(t_1\ t_2)[\![x/t']\!] = (t_1[\![x/t']\!])\ (t_2[\![x/t']\!])$;

4. $(\lambda x.t)[\![x/t']\!] = \lambda x.t$;
5. $(\lambda y.t)[\![x/t']\!] = \lambda y'.((t[y/y'])[\![x/t']\!])$ (variabila $y'$ trebuie să fie nouă/proaspătă/fresh, adică să nu mai apară în altă parte în termenii cu care lucrez).

Observați că singurul caz care s-a schimbat este ultimul caz, în care înainte de a înainta recursiv în termen pentru a substitui $x$ cu $t'$, variabila legată $y$ este înlocuită cu o variabilă nouă $y'$, folosind substituția clasică.

Avantajul substituției care evită capturarea este că aplicând o astfel de substituție pe doi termeni alfa-echivalenți, vom obține cu siguranță doi termeni alfa-echivalenți.

Ultimul caz se poate optimiza, evitând redenumirea în cazurile în care nu este necesară, în felul următor:

5. $(\lambda y.t)[\![x/t']\!] = \lambda y.(t[\![x/t']\!])$, dacă $y \notin \mathit{free}(t')$;
6. $(\lambda y.t)[\![x/t']\!] = \lambda y'.((t[y/y'])[\![x/t']\!])$, dacă $y \in \mathit{free}(t')$ (variabila $y'$ trebuie să fie nouă/proaspătă/fresh, adică să nu mai apară în altă parte în termenii cu care lucrez).

# 7 Regula de calcul

În lambda-calcul, există o singură regulă care modelează un pas de calcul, regulă care se numește beta-reducere.

Regula de $\beta$-reducere este următoarea:

$$(\lambda x.t)\ t' \to_\beta t[\![x/t']\!].$$

Cu alte cuvinte, singura regulă de calcul este aplicarea unei funcții ($(\lambda x.t)$) pe un argument ($t'$). Observați folosirea substituției care evită capturarea.

Exemplu de calcul:

1. $(\lambda x.x)\ y \to_\beta y \not\to_\beta$;
2. $(\lambda x.\lambda y.x\ y)\ x'\ y' \to_\beta (\lambda y.x'\ y)\ y' \to_\beta x'\ y' \not\to_\beta$;
3. $(\lambda x.\lambda y.x\ y)\ (\lambda x.x)\ y' \to_\beta (\lambda y.(\lambda x.x)\ y)\ y' \to_\beta (\lambda x.x)\ y' \to_\beta y' \not\to_\beta$;;
4. $(\lambda x.(\lambda y.x\ y))\ (y\ y) \to_\beta \lambda y'.(y\ y)\ y' \not\to_\beta$;;
5. $x\ ((\lambda x.x)\ y) \to_\beta x\ y \not\to_\beta$;.

Observați cum a fost evitată capturarea în penultimul exemplu.

Există și calcule infinite:

$$(\lambda x.x\ x)\ (\lambda x.x\ x) \to_\beta$$
$$(\lambda x.x\ x)\ (\lambda x.x\ x) \to_\beta$$
$$\ldots$$

# 8 Codări Church

Deși aparent simplu, și cu o singură regulă de calcul (aplicarea unei funcții), lambda-calculul este un limbaj la fel de puternic ca orice alt limbaj. Din punct de vedere matematic, limbajul este Turing-complet, adică poate calcula orice funcție care poate fi calculată de o mașină Turing.

Pentru a arăta acest lucru, vom folosi un set de codări (encodings) inteligent proiectate, numite codări Church (Alonzo Church a creat calculul lambda).

## 8.1 Valori Booleene

Iată codările pentru valori booleene:

$$\begin{array}{rcl} TRUE & = & \lambda x.\lambda y.x \\ FALSE & = & \lambda x.\lambda y.y \end{array}$$

Valoarea "adevărat" va fi reprezentată de o funcție care primește două argumente și întoarce primul dintre ele, iar valoarea "fals" de o funcție cu două argumente care întoarce al doilea argument.

Operațiile booleene obișnuite sunt codate după cum urmează:

$$\begin{array}{rcl} AND & = & \lambda u.\lambda v.u\ v\ u \\ OR & = & \lambda u.\lambda v.u\ u\ v \\ NOT & = & \lambda u.u\ FALSE\ TRUE \end{array}$$

Putem verifica că într-adevăr termenii de mai sus se comportă conform așteptărilor:

$$\begin{array}{l} AND\ TRUE\ FALSE = \\ (\lambda u.\lambda v.u\ v\ u)\ TRUE\ FALSE \to_\beta \\ (\lambda v.TRUE\ v\ TRUE)\ FALSE \to_\beta \\ TRUE\ FALSE\ TRUE = \\ (\lambda x.\lambda y.x)\ FALSE\ TRUE \to_\beta \\ (\lambda y.FALSE)\ TRUE \to_\beta \\ FALSE. \end{array}$$

$$\begin{array}{l} AND\ TRUE\ TRUE = \\ (\lambda u.\lambda v.u\ v\ u)\ TRUE\ TRUE \to_\beta \\ (\lambda v.TRUE\ v\ TRUE)\ TRUE \to_\beta \\ TRUE\ TRUE\ TRUE = \\ (\lambda x.\lambda y.x)\ TRUE\ TRUE \to_\beta \\ (\lambda y.TRUE)\ TRUE \to_\beta \\ TRUE. \end{array}$$

$$\begin{array}{l} AND\ FALSE\ TRUE = \\ (\lambda u.\lambda v.u\ v\ u)\ FALSE\ TRUE \to_\beta \\ (\lambda v.FALSE\ v\ FALSE)\ TRUE \to_\beta \\ FALSE\ TRUE\ FALSE = \\ (\lambda x.\lambda y.y)\ TRUE\ FALSE \to_\beta \\ (\lambda y.y)\ FALSE \to_\beta \\ FALSE. \end{array}$$

$$\begin{array}{l} AND\ FALSE\ FALSE = \\ (\lambda u.\lambda v.u\ v\ u)\ FALSE\ FALSE \to_\beta \\ (\lambda v.FALSE\ v\ FALSE)\ FALSE \to_\beta \\ FALSE\ FALSE\ FALSE = \\ (\lambda x.\lambda y.y)\ FALSE\ FALSE \to_\beta \\ (\lambda y.y)\ FALSE \to_\beta \\ FALSE. \end{array}$$

Exercițiu: verificați comportamentul funcțiilor *OR* și *NOT*.

## 8.2 Numere Naturale

Numele naturale pot fi codate după cum urmează:

$$\begin{aligned} \mathit{0} &= \lambda f.\lambda x.x \\ \mathit{1} &= \lambda f.\lambda x.f\ x \\ \mathit{2} &= \lambda f.\lambda x.f\ f\ x \\ \mathit{3} &= \lambda f.\lambda x.f\ f\ f\ x \\ &\dots \end{aligned}$$

Cu alte cuvinte, un număr natural $n$ va fi reprezentat printr-o funcție care primește două argumente: f și x și aplică f de $n$ ori pe x.

Funcția succesor este foarte simplu de scris ca lambda-termen:

$$\mathit{SUCC} = \lambda n.\lambda f.\lambda x.((n\ f)\ (f\ x)).$$

Să calculăm $\mathit{SUCC}\ 2$:

$\mathit{SUCC}\ 2 =$
$\lambda n.\lambda f.\lambda x.((n\ f)\ (f\ x))\ 2 \rightarrow_\beta$
$\lambda f.\lambda x.((2\ f)\ (f\ x)) =$
$\lambda f.\lambda x.(((\lambda f.\lambda x.f\ f\ x)\ f)\ (f\ x)) \rightarrow_\beta$
$\lambda f.\lambda x.((\lambda x.f\ f\ x)\ (f\ x)) \rightarrow_\beta$
$\lambda f.\lambda x.(f\ f\ f\ x) =$
3.

Funcția de adunare a două numere poate fi reprezentată ca lambda-termen după cum urmează:

$$\mathit{PLUS} = \lambda n.\lambda m.\lambda f.\lambda x.m\ f\ (n\ f\ x).$$

Exercițiu: calculați $\mathit{PLUS}\ 2\ 3$.