Fundamentele limbajelor de programare
CURS 7

Andrei Sipoș[1]

Facultatea de Matematică și Informatică, DL Info, Anul II
Semestrul II, 2025/2026

[1] Curs realizat împreună cu Traian Florin Șerbănuță.

Programare funcțională

Lambda-calcul

Limbajele de programare funcționale (Haskell, ML etc.) sunt caracterizate, în primul rând, prin faptul că funcțiile sunt „cetățeni de prim rang" ("first-class citizens"), ele fiind obiecte ca oricare altele (cum sunt numerele sau valorile booleene), iar această trăsătură derivă direct din limbajul din care au fost ele inspirate inițial, anume **lambda-calculul (λ-calculul)** lui Church, pomenit în Introducerea istorică.

Lambda-calculul a fost conceput, după cum am zis, pentru a reprezenta un model de calcul universal, capabil să reprezinte toate funcțiile informal calculabile, și de aceea a și fost gândit ca fiind un calcul al funcțiilor. El este prezent azi în felurite variante, în special cu tipuri, având diverse restricții și expansiuni. Noi vom începe cu prezentarea calculului original, fără tipuri.

Lambda-calcul fără tipuri

Fixăm o mulțime numărabilă de variabile $V = \{x_n \mid n \in \mathbb{N}\}$.

Un λ-**termen (fără tipuri)** are exact una dintre următoarele forme:

- o **variabilă** $x \in V$;
- o **(λ)-abstracțiune**: dacă $x \in V$ și M este un λ-termen, atunci $\lambda x.M$ este un λ-termen;
- o **aplicație**: dacă M și N sunt λ-termeni, atunci MN este un λ-termen.

Cum arată principiile de inducție/recursie corespunzătoare?

Variabile și variabile libere

Putem defini recursiv mulțimea variabilelor unui termen prin clauzele:

- $Var(v) = \{v\}$;
- $Var(\lambda x.M) = Var(M) \cup \{x\}$;
- $Var(MN) = Var(M) \cup Var(N)$;

iar pe cea a variabilelor libere prin:

- $FV(v) = \{v\}$;
- $FV(\lambda x.M) = FV(M) \setminus \{x\}$;
- $FV(MN) = FV(M) \cup FV(N)$.

Interpretarea informală

După cum am zis, λ-termenii au fost gândiți ca reprezentând funcții. Mai exact (fixând x, y, $z \in V$ distincte două câte două):

- Un termen de forma $\lambda x.M$ este gândit ca reprezentând funcția care duce pe x în M (M fiind un termen în a cărui componență poate sau nu să apară variabila x).
  De exemplu: $\lambda x.x$ ar reprezenta funcția identitate, $\lambda x.y$ ar reprezenta o funcție constant egală cu y.
- Un termen de forma MN reprezintă rezultatul aplicării „funcției" M pe „argumentul" N.
  De exemplu: am vrea ca $(\lambda x.x)z$ să reprezinte z, iar $(\lambda x.y)z$ să reprezinte y.

Remarcăm că aceste interpretări sunt aici pur informale: a le face riguroase a fost mult timp o problemă aproape insurmontabilă. Situația devine mai ușoară dacă nu ne propunem să formalizăm termenii ca funcții („semantică denotațională"), ci doar să stabilim regulile prin care ei sunt manipulați („semantică operațională").

Spre substituție

De exemplu, când spunem că $(\lambda x.x)z$ am vrea să reprezinte z, sugerăm că x-ul din „corpul funcției" am vrea să fie substituit cu z. Pentru aceasta, avem nevoie de o definiție a substituției. Nu putem substitui **naiv** variabilele cu λ-termeni din aceleași considerente ca la logica de ordinul I: am putea să ne trezim cu urmări nedorite, de exemplu, dacă în λ-termenul

$$\lambda x.y,$$

reprezentând funcția „constant egală cu y", substituim fără atenție y cu x, ajungem la λ-termenul

$$\lambda x.x,$$

care reprezintă o funcție identitate (variabila x fiind „capturată accidental" de către λx). Or, aceasta contravine intuiției care ne spune că o funcție compusă cu una constantă nu poate fi neconstantă.

Spre substituție

Observăm următorul fapt: termeni ca $\lambda x.x$ și $\lambda z.z$ am dori să denote aceeași funcție, funcția identitate; la fel și $\lambda x.y$ și $\lambda z.y$ aceeași funcție, funcția constant egală cu y. Așadar, vom transforma întâi $\lambda x.y$ în

$$\lambda z.y,$$

pentru a putea substitui apoi y cu x, obținând

$$\lambda z.x,$$

care este tot o funcție constantă.

Practic, ideea este că denumirile variabilelor legate nu contează, atâta timp cât ele sunt folosite consecvent: de aceea, ele se pot și substitui una cu alta atâta timp cât și substituția este consecventă. Dat fiind că în acest principiu se amintește de substituție, el se va putea formaliza abia după definirea substituției. Totuși, acea parte a sa care este relevantă pentru definirea substituției poate fi inclusă în definiție, folosind recursivitatea.

Definirea substituției

Pentru orice λ-termeni M, N și orice $x \in V$, vom defini termenul $M[x := N]$, reprezentând M în care x a fost înlocuit cu N. O vom face recursiv, în felul următor (unde x, $y \in V$ cu $x \neq y$, iar N, P, Q sunt λ-termeni):

- $x[x := N] := N$;
- $x[y := N] := x$;
- $(PQ)[x := N] := (P[x := N])(Q[x := N])$;
- $(\lambda x.P)[x := N] := \lambda x.P$;
- $(\lambda y.P)[x := N] := \lambda y.(P[x := N])$, dacă $y \notin FV(N)$;
- $(\lambda y.P)[x := N] := \lambda z.(P[y := z][x := N])$, dacă $y \in FV(N)$, unde z este variabila de indice minim care nu aparține lui $Var(\lambda x.(NP))$, caz care corespunde fenomenului prezentat mai devreme.

Exemple

Care sunt următorii λ-termeni (presupunem u, v, w, x, y, $z \in V$, distincte două câte două)?

- $(\lambda y.(x(\lambda w.((vw)x))))[x := uv]$;
- $(\lambda y.(x(\lambda x.x)))[x := \lambda y.(xy)]$;
- $(y(\lambda v.(xv)))[x := \lambda y.(vy)]$;
- $(\lambda x.(zy))[x := uv]$.

Alpha-echivalență

În acest moment, putem formaliza intuiția de mai devreme legată de substituția variabilelor legate. Numim α-echivalență și o notăm cu $\equiv_\alpha$ cea mai mică relație de echivalență $\equiv$ pe λ-termeni care satisface următoarele:

- pentru orice x, $y \in V$ și orice λ-termen M cu $y \notin FV(M)$, $\lambda x.M \equiv \lambda y.(M[x := y])$;
- pentru orice $x \in V$ și orice λ-termeni M, N cu $M \equiv N$, avem $\lambda x.M \equiv \lambda x.N$;
- pentru orice λ-termeni M, N, P cu $M \equiv N$, avem $MP \equiv NP$ și $PM \equiv PN$.

Spre reducție

Am spus mai devreme că $(\lambda x.x)z$ am vrea să reprezinte z, iar pentru aceasta am introdus o definiție a substituției astfel încât $x[x := z] = z$. Totuși, mai trebuie să spunem și de ce putem face trecerea

$$(\lambda x.x)z \to x[x := z]$$

sau, în celălalt exemplu,

$$(\lambda x.y)z \to y[x := z]$$

și, în general,

$$(\lambda x.M)N \to M[x := N].$$

Pentru aceasta, vom introduce o nouă relație pe λ-termeni, care va reprezenta această procedură de reducție.

Beta-reducție

Numim β-reducție și o notăm cu $\rightarrow_\beta$ cea mai mică relație $\rightarrow$ pe λ-termeni care satisface următoarele, pentru orice λ-termeni M, N, P și orice $x \in V$:

- $(\lambda x.M)N \rightarrow M[x := N]$;
- dacă $M \rightarrow N$, atunci $\lambda x.M \rightarrow \lambda x.N$, $MP \rightarrow NP$ și $PM \rightarrow PN$.

Notăm cu $\rightarrow_\beta^*$ închiderea reflexiv-tranzitivă a lui $\rightarrow_\beta$.

Un λ-termen M se numește **formă normală** dacă nu există N cu $M \rightarrow_\beta N$. Dacă M și N sunt λ-termeni, N se numește **formă normală a lui** M dacă $M \rightarrow_\beta^* N$ și N este formă normală.

Normalizare și confluență

Dacă notăm $I := \lambda x.x$, $\omega := \lambda x.(xx)$ și $\Omega := \omega\omega$, se observă că, pentru orice M,

$$IM = (\lambda x.x)M \to_\beta x[x := M] = M$$

și

$$\omega M = (\lambda x.(xx))M \to_\beta (xx)[x := M] = MM.$$

Așadar, $\Omega = \omega\omega \to \omega\omega = \Omega$, deci există șiruri infinite de β-reducții. Aceasta face ca λ-calculul fără tipuri să nu fie **normalizant**.

El are, în schimb, o proprietate pozitivă, anume aceea de **confluență** (modulo α-echivalență): pentru orice M, N_1, N_2 cu $M \to_\beta^* N_1$ și $M \to_\beta^* N_2$, există P_1, P_2 cu $N_1 \to_\beta^* P_1$, $N_2 \to_\beta^* P_2$ și $P_1 \equiv_\alpha P_2$.

Valori booleene

Am spus că λ-calculul se vrea să fie un model de calcul, așadar ne punem problema de a reprezenta date și operații pe ele în el. Cel mai simplu tip de date este cel boolean. Dacă notăm cu **true** λ-termenul $\lambda x.\lambda y.x$, iar cu **false** λ-termenul $\lambda x.\lambda y.y$, atunci ideea de **if** M **then** P **else** Q se poate reprezenta prin $(MP)Q$, având în vedere că, pentru orice λ-termeni M, P, și Q, avem că, dacă $M \to_\beta^*$ **true**, atunci

$$(MP)Q \to_\beta^* P,$$

iar, dacă $M \to_\beta^*$ **false**, atunci

$$(MP)Q \to_\beta^* Q.$$

Numere naturale

Trecem acum la întrebarea: cum putem reprezenta numere naturale? Dat fiind că λ-calculul este un calcul al funcțiilor, întrebarea se reduce la: cum putem privi un număr ca o funcție care acționează asupra altor funcții? Un răspuns posibil, dat de Church, este următorul: un $n \in \mathbb{N}$ este privit ca funcția

$$f \mapsto f^{(n)},$$

unde notația $f^{(n)}$ reprezintă compunerea lui f cu ea însăși de n ori. Așadar, numărul n va fi reprezentat de λ-termenul $\lambda f.\lambda x.(f^{(n)}x)$.

Numere naturale

Riguros vorbind, luăm f, $x \in V$ cu $f \neq x$, punem $D_0 := x$ și, pentru orice $n \in \mathbb{N}$, $D_{n+1} := fD_n$. Punem, apoi, pentru orice $n \in \mathbb{N}$, $C_n := \lambda f.\lambda x.D_n$ și îl numim **numeralul Church** asociat lui n.

Putem acum defini un „predicat" care verifică dacă un număr este zero, prin:

$$\textbf{iszero} := \lambda n.\lambda x.\lambda y.((n(\lambda z.y))x).$$

Observăm că, de exemplu,

$$\begin{aligned}\textbf{iszero}\ C_0 = \textbf{iszero}\ (\lambda f.\lambda x.x) &\to_\beta \lambda x.\lambda y.(((\lambda f.\lambda x.x)(\lambda z.y))x) \\ &\to_\beta \lambda x.\lambda y.((\lambda x.x)x) \to_\beta \lambda x.\lambda y.x = \textbf{true}.\end{aligned}$$

Similar, $\textbf{iszero}\ C_1 \to_\beta^* \textbf{false}$.

Succesorul

Dat fiind că, intuitiv vorbind, pentru orice n,

$$C_n(f)(x) = f^{(n)}(x)$$

și

$$C_{n+1}(f)(x) = f^{(n+1)}(x) = f(C_n(f)(x)),$$

putem defini un λ-termen care să reprezinte funcția succesor, prin

$$\mathbf{succ} := \lambda n.\lambda f.\lambda x.(f((nf)x)).$$

Testați comportamentul acestui λ-termen!

Alte operații pe numere

Pentru adunare, observăm că, la fel ca înainte, pentru orice n, $m \in \mathbb{N}$, intuitiv vorbind

$$C_{n+m}(f)(x) = f^{(n+m)}(x) = f^{(n)}(f^{(m)}(x)) = C_n(f)(C_m(f)(x)),$$

ceea ce ne conduce la definiția:

$$\textbf{add} := \lambda n.\lambda m.\lambda f.\lambda x.((nf)((mf)x)).$$

Similar, putem defini:

$$\textbf{mul} := \lambda n.\lambda m.\lambda f.(n(mf)).$$

$$\textbf{pow} := \lambda b.\lambda e.(eb).$$

$$\textbf{pred} := \lambda n.\lambda f.\lambda x.(((n(\lambda g.\lambda h.(h(gf))))(\lambda u.x))(\lambda u.u)).$$

Perechi

Putem reprezenta și perechi în λ-calcul:

$$\textbf{pair} := \lambda x.\lambda y.\lambda f.(fxy).$$

$$\textbf{fst} := \lambda p.(p\ \textbf{true}).$$

$$\textbf{snd} := \lambda p.(p\ \textbf{false}).$$

Aceasta ne conduce la o reprezentare mai transparentă a funcției predecesor:

$$\textbf{aux} := \lambda x.(\textbf{pair}\ (\textbf{snd}\ x)(\textbf{succ}\ (\textbf{snd}\ x))).$$

$$\textbf{predp} := \lambda n.(\textbf{fst}\ (n\ \textbf{aux}\ (\textbf{pair}\ C_0\ C_0))).$$

Recursivitate

Cum definim funcții recursive? Spre exemplu, funcția factorial ar satisface intuitiv ecuația

$$\textbf{fact} = \lambda n.(((\textbf{iszero}\ n)C_1)((\textbf{mul}\ n)(\textbf{fact}(\textbf{pred}\ n)))),$$

sau, altfel spus,

$$\textbf{fact} = (\lambda f.(\lambda n.(((\textbf{iszero}\ n)C_1)((\textbf{mul}\ n)(f(\textbf{pred}\ n))))))\textbf{fact}.$$

Dacă am avea la dispoziție un **operator de punct fix**, adică un λ-termen Y astfel încât, pentru orice M, $YM \to_\beta^* M(YM)$, atunci am putea, ca în Laboratorul 3, să definim

$$\textbf{fact} := Y(\lambda f.(\lambda n.(((\textbf{iszero}\ n)C_1)((\textbf{mul}\ n)(f(\textbf{pred}\ n)))))).$$

Puncte fixe

Dacă punem $U := \lambda u.\lambda x.(x((uu)x))$, atunci $Y := UU$ este un asemenea operator de punct fix, având în vedere că, pentru orice M, avem

$$\begin{aligned} YM &= (UU)M \\ &\to_\beta ((\lambda x.(x((uu)x)))[u := U])M \\ &= (\lambda x.(x((UU)x)))M \\ &= (\lambda x.(x(Yx)))M \\ &\to_\beta (x(Yx))[x := M] \\ &= M(YM). \end{aligned}$$

Această soluție a fost dată de Alan Turing. Altă soluție, dată de Haskell Curry, este $Y := \lambda u.((\lambda x.u(xx))(\lambda x.u(xx)))$.

Funcții reprezentabile

În general, spunem, pentru orice $k \in \mathbb{N}$, că o funcție $f : \mathbb{N}^k \to \mathbb{N} \cup \{\bot\}$ este λ-**reprezentabilă** dacă există un λ-termen M astfel încât, pentru orice $n_1, \ldots, n_k \in \mathbb{N}$, dacă $f(n_1, \ldots, n_k) \in \mathbb{N}$, atunci

$$MC_{n_1} \ldots C_{n_k} \to_\beta^* C_{f(n_1,\ldots,n_k)},$$

iar, dacă $f(n_1, \ldots, n_k) = \bot$, atunci

$$MC_{n_1} \ldots C_{n_k}$$

nu are formă normală.

Calculabilitate

Se poate demonstra că aceste funcții coincid exact cu funcțiile calculabile de o mașină Turing (și, în general, de orice alt model de calcul universal care a fost conceput de om), ceea ce dă crezare **tezei Church-Turing**, care spune că această clasă capturează exact clasa funcțiilor calculabile la modul informal (mai multe despre aceasta se va spune la cursul „Calculabilitate și complexitate").

Totuși, vedem că acest λ-calcul fără tipuri cuprinde în el și termeni care nu au formă normală, și de aceea în definirea funcțiilor reprezentabile am inclus și funcții parțiale. Există, oare, un mod de a ne restrânge la funcțiile calculabile care sunt totale? Mai multe despre aceasta vom vedea în cursul următor.