Programare funcțională

Introducere în Lambda-calcul — Totul din (aproape) nimic

Ioana Leuștean
Traian Florin Șerbănuță

Departamentul de Informatică, FMI, UB
ioana@fmi.unibuc.ro
traian.serbanuta@unibuc.ro

λ-calcul pentru toate problemele noastre

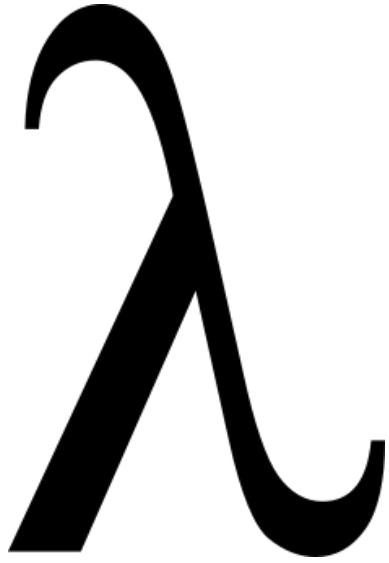

Figura: All you need is Lambda

- Model de calcul introdus de Alonzo Church
- Formalizează conceptul de computabilitate
  - Ce clase de probleme pot fi rezolvate cu un calculator?
- Baza programării funcționale
  - toate limbajele funcționale sunt la bază un λ-calcul

Ce este programarea funcțională

- Paradigmă de programare ce folosește funcții modelate după funcțiile din matematică
- Programele se obțin ca o combinație de expresii
- Expresiile pot fi valori concrete, variable și funcții
- Funcțiile sunt expresii ce pot fi aplicate unor intrări
  - În urma aplicării, o funcție e redusă sau evaluată
- Funcțiile sunt valori (first-class citizens)
  - pot fi folosite ca argumente pentru alte funcții

Ce este programarea funcțională

- Paradigmă de programare ce folosește funcții modelate după funcțiile din matematică
- Programele se obțin ca o combinație de expresii
- Expresiile pot fi valori concrete, variable și funcții
- Funcțiile sunt expresii ce pot fi aplicate unor intrări
  - În urma aplicării, o funcție e redusă sau evaluată
- Funcțiile sunt valori (first-class citizens)
  - pot fi folosite ca argumente pentru alte funcții

Puritate

- Toate limbajele funcționale sunt bazate pe lambda-calcul
- Unele limbaje încorporează și lucruri ce nu sunt reprezentabile în lambda-calcul
- Haskell e un limbaj pur, pentru că nu face acest compromis

Ce este o funcție

- Relație între două mulțimi (de intrare și de ieșire)
- Asociază fiecărei intrări exact o ieșire
  - transparență referențială: ieșirea e unic determinată de intrare

Definirea și evaluarea funcțiilor

- Exemplu definire: $f(x) = x + 1$
  - o funcție numită f
  - care dată fiind o valoare de intrare, să zicem x,
  - obține valoarea de ieșire conform expresiei $x + 1$
- Exemplu evaluare:
  $f(1) = 1 + 1$ (substitui x cu 1 in $x + 1$) $= 2$ (simplific)

Structura λ-expresiilor

O expresie este definită recursiv astfel:

- este o variabilă (un identificator)
  x
- se obține prin abstractizarea unei variabile x într-o altă expresie e
  $\lambda x.e$ exemplu: $\lambda x.x$
- se obține prin aplicarea unei expresii e_1 asupra alteia e_2
  $e_1\ e_2$ exemplu: $(\lambda x.x)y$

Structura λ-expresiilor

O expresie este definită recursiv astfel:

- este o variabilă (un identificator)
  x
- se obține prin abstractizarea unei variabile x într-o altă expresie e
  $\lambda x.e$ exemplu: $\lambda x.x$
- se obține prin aplicarea unei expresii e_1 asupra alteia e_2
  $e_1\ e_2$ exemplu: $(\lambda x.x)y$

Operația de abstractizare $\lambda x.e$

- reprezintă o funcție anonimă
- constă din două părți: antetul $\lambda x.$ și corpul e
- variabila x din antet este parametrul funcției
  - leagă aparițiile variabilei x în e (ca un cuantificator)
  - Exemplu: $\lambda x.xy$ — x e legată, y e liberă
- Corpul funcției reprezintă expresia care definește funcția

α-echivalență

- Redenumirea unui parametru și a tuturor aparițiilor sale legate
  - Exemplu: $\lambda x.x \equiv_\alpha \lambda y.y \equiv_\alpha \lambda a.a$
  - Asemanator cu: $f(x) = x$ vs $f(y) = y$ vs $f(a) = a$
- Numele asociat parametrului e pur formal
  - E necesar doar ca să îl pot recunoaște în corpul funcției
  - Există reprezentări fără variabile (e.g. indecși de Bruijn)
- α-echivalența redenumește doar aparițiile legate ale argumentului
  - Exemple:
    $(\lambda x.x)x \not\equiv_\alpha (\lambda y.y)y$
    $(\lambda x.x)x \equiv_\alpha (\lambda y.y)x$

β-reducție

Cum aplicăm o funcție (anonimă) unui argument?

Înlocuim aplicația cu corpul funcției în care substituim aparițiile legate ale parametrului cu argumentul dat.

$$(\lambda x.e)e' \rightarrow_\beta e[x := e']$$

Comparați cu funcțiile ne-anonime:

Dacă $f(x) = e$, atunci $f(e') = e[x := e']$

β-reducție

Cum aplicăm o funcție (anonimă) unui argument?

Înlocuim aplicația cu corpul funcției în care substituim aparițiile legate ale parametrului cu argumentul dat.

$$(\lambda x.e)e' \rightarrow_\beta e[x := e']$$

Exemple

$$(\lambda x.x)y \rightarrow_\beta x[x := y] = y$$

$$(\lambda x.x\ x)\lambda x.x \rightarrow_\beta x\ x[x := \lambda x.x] = (\lambda x.x)\lambda x.x \rightarrow_\beta x[x := \lambda x.x] = \lambda x.x$$

Alte exemple

Aplicarea funcțiilor se grupează la stânga

$$(\lambda x.x)(\lambda y.y)z = ((\lambda x.x)(\lambda y.y))z$$

Alte exemple

Aplicarea funcțiilor se grupează la stânga

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

Funcție cu variabile libere

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

lambda are are prioritate foarte mică

$\lambda x.x\ \lambda x.x = \lambda x.(x\ (\lambda x.x))$

Mai multe argumente

- Funcțiile anonime au un singur parametru
  - și pot fi aplicate unui singur argument
- Simulăm mai multe argumente prin abstractizare repetată
  Exemplu: $\lambda x.\lambda y.x\ y$
  - Citim: primește ca argumente x și y și aplică pe x lui y
  - De fapt e o funcție de x care în urma aplicării dă o funcție de y
  - Procesul se numește Currying (de la Haskell Curry)
  - Pentru simplificarea notației, scriem $\lambda x\ y.x\ y$ în loc de $\lambda x.\lambda y.x\ y$

Exemplu de evaluare

$$(\lambda x\ y.x\ y)(\lambda z.a)1 = (\lambda x.(\lambda y.x\ y))(\lambda z.a)1 = ((\lambda x.(\lambda y.x\ y))(\lambda z.a))1$$

Mai multe argumente

- Funcțiile anonime au un singur parametru
  - și pot fi aplicate unui singur argument
- Simulăm mai multe argumente prin abstractizare repetată
  Exemplu: $\lambda x.\lambda y.x\ y$
  - Citim: primește ca argumente x și y și aplică pe x lui y
  - De fapt e o funcție de x care în urma aplicării dă o funcție de y
  - Procesul se numește Currying (de la Haskell Curry)
  - Pentru simplificarea notației, scriem $\lambda x\ y.x\ y$ în loc de $\lambda x.\lambda y.x\ y$

Exemplu de evaluare

$(\lambda x\ y.x\ y)(\lambda z.a)1 = (\lambda x.(\lambda y.x\ y))(\lambda z.a)1 = ((\lambda x.(\lambda y.x\ y))(\lambda z.a))1 \rightarrow_\beta ((\lambda y.x\ y)[x := \lambda z.a])1 = (\lambda y.(\lambda z.a)y)1$

Mai multe argumente

- Funcțiile anonime au un singur parametru
  - și pot fi aplicate unui singur argument
- Simulăm mai multe argumente prin abstractizare repetată
  Exemplu: $\lambda x.\lambda y.x\ y$
  - Citim: primește ca argumente x și y și aplică pe x lui y
  - De fapt e o funcție de x care în urma aplicării dă o funcție de y
  - Procesul se numește Currying (de la Haskell Curry)
  - Pentru simplificarea notației, scriem $\lambda x\ y.x\ y$ în loc de $\lambda x.\lambda y.x\ y$

Exemplu de evaluare

$(\lambda x\ y.x\ y)(\lambda z.a)1 = (\lambda x.(\lambda y.x\ y))(\lambda z.a)1 = ((\lambda x.(\lambda y.x\ y))(\lambda z.a))1 \rightarrow_\beta ((\lambda y.x\ y)[x := \lambda z.a])1 = (\lambda y.(\lambda z.a)y)1 \rightarrow_\beta ((\lambda z.a)y)[y := 1] = (\lambda z.a)1$

Mai multe argumente

- Funcțiile anonime au un singur parametru
  - și pot fi aplicate unui singur argument
- Simulăm mai multe argumente prin abstractizare repetată
  Exemplu: $\lambda x.\lambda y.x\ y$
  - Citim: primește ca argumente x și y și aplică pe x lui y
  - De fapt e o funcție de x care în urma aplicării dă o funcție de y
  - Procesul se numește Currying (de la Haskell Curry)
  - Pentru simplificarea notației, scriem $\lambda x\ y.x\ y$ în loc de $\lambda x.\lambda y.x\ y$

Exemplu de evaluare

$(\lambda x\ y.x\ y)(\lambda z.a)1 = (\lambda x.(\lambda y.x\ y))(\lambda z.a)1 = ((\lambda x.(\lambda y.x\ y))(\lambda z.a))1 \rightarrow_\beta$
$((\lambda y.x\ y)[x := \lambda z.a])1 = (\lambda y.(\lambda z.a)y)1 \rightarrow_\beta ((\lambda z.a)y)[y := 1] =$
$(\lambda z.a)1 \rightarrow_\beta a[z := y] = a$

Evaluarea ca simplificare

Forma normală β

- O λ-expresie este în forma normală β dacă regula β nu mai poate fi aplicată asupra ei.
- Corespunde ideii de expresie complet evaluată
- Evaluarea unei λ-expresii constă în simplificarea ei
  - folosind regula β până la ajungerea la o formă normală
- O λ-expresie nu poate avea mai multe forme normale
  - modulo α-conversie

Exemple

- $\lambda x.x$ este în formă normală
- $(\lambda x.x)z$ nu este în formă normală (z este)
- $z(\lambda x.x)$ este în formă normală

Combinatori

- Combinatorii sunt λ-expresii fără variabile libere
- Scopul lor este de a "combina" argumentele primite la intrare

Contraexemple

- $\lambda y.x$ — y e legată, dar x este liberă
- $\lambda x.x\ z$ — x e legată, dar z este liberă

Exemple distinse

- **combinatorul I**: funcția identitate $\lambda x.x$
- **combinatorul K**: proiecția stângă $\lambda x\ y.x$
- **combinatorul S**: substituție $\lambda x\ y\ z.xz(yz)$
- orice λ-expresie poate fi obținută dintr-o combinație de S,K, I
- Vă place Tetris? Încercați Combinatris

Divergență

- Nu toate λ-expresiile au formă normală
  - cele care au se numesc convergente
  - celelalte se numesc divergente
- Exemplu de expresie divergentă: **combinatorul** ω
  $(\lambda x.x\ x)(\lambda x.x\ x)$

Divergență

- Nu toate λ-expresiile au formă normală
  - cele care au se numesc convergente
  - celelalte se numesc divergente
- Exemplu de expresie divergentă: **combinatorul** ω
  $(\lambda x.x\ x)(\lambda x.x\ x) \rightarrow_\beta (x\ x)[x := \lambda x.x\ x] = (\lambda x.x\ x)(\lambda x.x\ x)$

Mai multe despre currying

Funcții în matematică

- Fie $f : A \times B \to C$ o funcție. În mod uzual scriem $f(x, y) = z$ unde $x \in A$, $y \in B$ și $z \in C$.
- Pentru $x \in A$ (arbitrar, fixat) definim $f_x : B \to C$, $f_x(y) = z$ dacă și numai dacă $f(x, y) = z$. Funcția f_x se obține prin aplicarea parțială a funcției f.

Funcții în matematică

- Fie $f : A \times B \to C$ o funcție. În mod uzual scriem $f(x, y) = z$ unde $x \in A$, $y \in B$ și $z \in C$.
- Pentru $x \in A$ (arbitrar, fixat) definim $f_x : B \to C$, $f_x(y) = z$ dacă și numai dacă $f(x, y) = z$.
  Funcția f_x se obține prin aplicarea parțială a funcției f.

  In mod similar putem defini *aplicarea parțială* pentru orice $y \in B$

  $f^y : A \to C$, $f^y(x) = z$ dacă și numai dacă $f(x, y) = z$.

Funcții în matematică

- Fie $f : A \times B \to C$ o funcție. În mod uzual scriem $f(x, y) = z$ unde $x \in A$, $y \in B$ și $z \in C$.
- Pentru $x \in A$ (arbitrar, fixat) definim $f_x : B \to C$, $f_x(y) = z$ dacă și numai dacă $f(x, y) = z$. Funcția f_x se obține prin aplicarea parțială a funcției f.
- Dacă notăm $B \to C \stackrel{not}{=} \{h : B \to C \mid h \text{ funcție}\}$ observăm că $f_x \in B \to C$ pentru orice $x \in A$.

Funcții în matematică

- Fie $f : A \times B \to C$ o funcție. În mod uzual scriem $f(x, y) = z$ unde $x \in A$, $y \in B$ și $z \in C$.
- Pentru $x \in A$ (arbitrar, fixat) definim $f_x : B \to C$, $f_x(y) = z$ dacă și numai dacă $f(x, y) = z$. Funcția f_x se obține prin aplicarea parțială a funcției f.
- Dacă notăm $B \to C \stackrel{not}{=} \{h : B \to C \mid h \text{ funcție}\}$ observăm că $f_x \in B \to C$ pentru orice $x \in A$.
- Asociem lui f funcția $cf : A \to (B \to C)$, $cf(x) = f_x$ Observăm că pentru fiecare element $x \in A$, funcția cf întoarce ca rezultat funcția $f_x \in B \to C$, adică $cf(x)(y) = z$ dacă și numai dacă $f(x, y) = z$

Funcții în matematică

- Fie $f : A \times B \to C$ o funcție. În mod uzual scriem $f(x, y) = z$ unde $x \in A$, $y \in B$ și $z \in C$.
- Pentru $x \in A$ (arbitrar, fixat) definim $f_x : B \to C$, $f_x(y) = z$ dacă și numai dacă $f(x, y) = z$. Funcția f_x se obține prin aplicarea parțială a funcției f.
- Dacă notăm $B \to C \stackrel{not}{=} \{h : B \to C \mid h \text{ funcție}\}$ observăm că $f_x \in B \to C$ pentru orice $x \in A$.
- Asociem lui f funcția $cf : A \to (B \to C)$, $cf(x) = f_x$. Observăm că pentru fiecare element $x \in A$, funcția cf întoarce ca rezultat funcția $f_x \in B \to C$, adică $cf(x)(y) = z$ dacă și numai dacă $f(x, y) = z$

Forma curry

Vom spune că funcția cf este *forma curry* a funcției f.

Currying

Teoremă

Mulțimile $(A \times B) \to C$ și $A \to (B \to C)$ sunt echipotente.

"Currying" este procedeul prin care o funcție cu mai multe argumente este transformată într-o funcție care are un singur argument și întoarce o altă funcție.

Currying in Haskell

Teoremă

Mulțimile $(A \times B) \to C$ și $A \to (B \to C)$ sunt echipotente.

Observație

Funcțiile curry și uncurry din Haskell stabilesc bijecția din teoremă:

```
curry :: ((a, b) -> c) -> a -> b -> c
curry f a b = f (a, b)

uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (a,b) = f a b
```

Currying în Haskell

- In Haskell toate funcțiile sunt forma **curry**, deci au un singur argument.
- Operatorul $\rightarrow$ pe tipuri este asociativ la dreapta, adică tipul $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n$ îl gândim ca $a_1 \rightarrow (a_2 \rightarrow \cdots (a_{n-1} \rightarrow a_n) \cdots)$.
- Aplicarea funcțiilor este asociativă la stânga, adică expresia $f\ x_1 \cdots x_n$ o gândim ca $(\cdots ((f\ x_1)\ x_2) \cdots x_n)$.