Fundamentele limbajelor de programare

Programare funcțională. Lambda-calcul cu tipuri simple.

Traian Florin Șerbănuță și Andrei Sipoș

Facultatea de Matematică și Informatică, DL Info

Anul II, Semestrul II, 2024/2025

Motivație pentru tipuri

Proprietăți negative ale lambda calculului fără tipuri:

- Aplicări de forma $x\,x$ sau $M\,M$ sunt pemise, deși sunt contraintuitive.
- Existența formelor normale pentru λ-termeni nu este garantată și putem avea „calcule infinite" nedorite
- Orice λ-termen are un punct fix ceea ce nu este în armonie cu ceea ce știam despre funcții oarecare

Vrem să eliminăm aceste proprietăți negative, păstrându-le pe cele pozitive.

Proprietățile negative sunt eliminate prin adăugarea de **tipuri** ceea ce induce restricțiile dorite pe termeni.

Secțiunea 1

Tipuri simple - noțiuni de bază

Tipuri simple

Fixăm o mulțime de „tipuri de bază”, notată cu T_0. De obicei notăm aceste tipuri cu $\alpha, \beta, \gamma, \ldots$

Mulțimea tipurilor simple (numite **tipuri**, de acum încolo), notată cu T, va fi cea mai mică mulțime care:

- conține pe T_0;
- pentru orice ρ, $\tau \in T$, avem $\rho \to \tau \in T$ (cu semnificația: „tipul funcțiilor de la ρ la τ”).

Tipuri simple – notații

Mulțimea tipurilor simple $T = T_0 \mid T \to T$

Exemple de tipuri simple:

- γ
- $(\beta \to \gamma)$
- $((\gamma \to \alpha) \to (\alpha \to (\beta \to \gamma)))$

În tipurile „săgeată”, parantezele exterioare pot fi omise.

Parantezele în tipurile săgeată sunt asociative la dreapta.
De exemplu,

- $\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4$ este abreviere pentru $(\alpha_1 \to (\alpha_2 \to (\alpha_3 \to \alpha_4)))$
- $x_1\, x_2\, x_3\, x_4$ este abreviere pentru $(((x_1\, x_2)\, x_3)\, x_4)$

Tipurile, intuitiv

Cum funcționează, intuitiv, tipurile?

Variabile În primul rând, variabilele „vor avea” tipuri (intenționat păstrăm ambiguitatea asupra sintagmei, vom vedea imediat de ce).

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

Termeni și tipuri – Exemple

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.

Discuție despre asocitativitate

Asociativitatea la dreapta pentru tipurile săgeată vs. asociativitatea la stânga pentru aplicare:

- Să presupunem că $f : \rho \to (\sigma \to \tau)$, $x : \rho$ și $y : \sigma$.
- Atunci $f\,x : \sigma \to \tau$ și $(f\,x)\,y : \tau$.
- Folosind ambele convenții pentru asociativitate pentru a elimina parantezele, avem

$$\frac{f : \rho \to \sigma \to \tau \qquad f\,x\,y : \tau}{}$$

Convențiile pentru asociativitate sunt în armonie una cu cealaltă.

Church-typing vs. Curry-typing

A găsi tipul unui termen începe cu a găsi tipurile pentru variabile.
Există două metode prin care putem asocia tipuri variabilelor.

Asociere explicită *(Church-typing)*.

- Constă în prescrierea unui unic tip pentru fiecare variabilă, la introducerea acesteia.
- Tipurile termenilor mai complecși se obțin natural, ținând cont de convențiile pentru aplicare și abstractizare.

Asociere implicită *(Curry-typing)*.

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

Secțiunea 2

Tipuri à la Church

Tipuri à la Church

În acest sistem, termenii sunt din start definiți ca având tipuri:

- avem câte o mulțime numărabilă de variabile pentru fiecare tip (aceste mulțimi fiind disjuncte două câte două);
- dacă x este o variabilă de tip ρ și M este un termen de tip τ, atunci $\lambda x.M$ este un termen de tip $\rho \to \tau$;
- dacă M și N sunt termeni, iar ρ și τ sunt tipuri astfel încât M este de tip $\rho \to \tau$, iar N este de tip ρ, atunci $M\ N$ este un termen de tip τ.

De exemplu, dacă α, β, $\gamma \in T$, iar x, y, z și u sunt variabile de tip $\alpha \to \alpha$, $(\alpha \to \alpha) \to \beta$, β și γ, respectiv, atunci $(\lambda z.(\lambda u.z))(yx)$ este un termen de tip $\gamma \to \beta$.

Exemplu

Aplicare Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.

Abstractizare Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

Vrem să calculăm tipul expresiei $(\lambda zu.\, z)\,(y\,x)$ știind că

1. $x : \alpha \to \alpha$
2. $y : (\alpha \to \alpha) \to \beta$
3. $z : \beta$
4. $u : \gamma$

Exemplu

Aplicare Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.

Abstractizare Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

Vrem să calculăm tipul expresiei $(\lambda zu.\, z)\,(y\,x)$ știind că

1. $x : \alpha \to \alpha$
2. $y : (\alpha \to \alpha) \to \beta$
3. $z : \beta$
4. $u : \gamma$

Din (2) și (1), prin aplicare obținem (5): $y\,x : \beta$.

Din (4) și (3), prin abstractizare obținem (6): $\lambda u.\, z : \gamma \to \beta$.

Din (3) și (6), prin abstractizare obținem (7): $\lambda zu.\, z : \beta \to \gamma \to \beta$.
Nu uitați că $\beta \to \gamma \to \beta$ înseamnă $\beta \to (\gamma \to \beta)$.

Atunci, din (7) și (5), prin aplicare, avem $(\lambda zu.\, z)\,(y\,x) : \gamma \to \beta$.

Substituție și reducție

Substituția se definește întocmai ca la λ-calculul fără tipuri, dar având grijă la tipuri. De exemplu, dacă $\sigma, \tau \in T$, t este un termen de tip τ, x este o variabilă de tip σ, iar u un termen de tip σ, atunci va avea sens să definim

$$t[x := u],$$

care va fi, apoi, un termen de tip τ.

Similar, relația de β-reducție se definește identic, având grijă, de exemplu, ca, în clauza fundamentală

$$(\lambda x.t)u \to t[x := u],$$

obiectele să aibă tipurile potrivite.

Normalizare și confluență

Se poate arăta (și chiar vom arăta într-un curs viitor) că acest λ-calcul cu tipuri simple nu mai posedă rescrieri infinite (altfel spus, are proprietatea de **normalizare tare**), i.e. nu există o familie de termeni $(M_n)_{n\in\mathbb{N}}$ astfel încât pentru orice $n \in \mathbb{N}$, $M_n \to_\beta M_{n+1}$.

În plus, el păstrează proprietatea de **confluență** a λ-calculului fără tipuri.

Secțiunea 3

Tipuri à la Curry

Tipuri à la Curry

Un al doilea mod de a gândi tipurile este sistemul de tipuri à la Curry. Aici, termenii sunt fără tipuri, mai exact ei sunt exact termeni din λ-calculul fără tipuri, iar lor li se alocă tipuri de către un sistem de deducție. Mai precis, vom emite judecăți de forma

$$\Gamma \vdash M : \tau,$$

însemnând: în contextul Γ i se alocă termenului M tipul τ. Un context va fi o mulțime finită de obiecte de forma $x : \sigma$, cu x variabilă și $\sigma \in T$ (simbolizând faptul că variabilei x i se alocă tipul σ), astfel încât orice variabilă apare cu cel mult un tip.

Reguli de deducție

Regulile de deducție vor fi următoarele:

$$\frac{}{\Gamma \cup \{x : \sigma\} \vdash x : \sigma}$$

$$\frac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}$$

Încercați să deduceți tipul termenului „de mai devreme” $(\lambda z.(\lambda u.z))(yx)$ folosind aceste reguli. Acest exemplu ne arată că, într-un anume fel, cele două moduri de a gândi tipurile sunt echivalente.

Exemplu

Aplicare Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.

Abstractizare Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

Considerăm termenul de mai devreme $M = (\lambda zu.\, z)\,(y\,x)$.

Putem să "ghicim" tipurile variabilelor astfel încât M să *aibă tip*?

Exemplu

Aplicare Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.

Abstractizare Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

Considerăm termenul de mai devreme $M = (\lambda zu.\, z)\,(y\,x)$.

Putem să "ghicim" tipurile variabilelor astfel încât M să *aibă tip*?

- Observăm că M este o aplicare a lui $\lambda zu.\, z$ termenului $y\,x$.
- Atunci $\lambda zu.\, z$ trebuie să aibă un tip săgeată, de exemplu $\lambda zu.\, z : A \to B$, și $y\,x$ să se potrivească, adică $y\,x : A$.
- În acest caz, avem $M : B$.

Exemplu (cont.)

Aplicare Dacă $M : \sigma \to \tau$ și $N : \sigma$, atunci $M\,N : \tau$.

Abstractizare Dacă $x : \sigma$ și $M : \tau$, atunci $\lambda x.\, M : \sigma \to \tau$.

Știm $M = (\lambda zu.\, z)\,(y\,x)$ și am dedus până acum:

$\lambda zu.\, z : A \to B$ $y\,x : A$ $M : B$

- Faptul că $\lambda zu.\, z : A \to B$ implică că $z : A$ și $\lambda u.\, z : B$.
- Deducem că B este tipul unei abstractizări, deci $B \equiv C \to D$, și obținem că $u : C$ și $z : D$.
- Pe de altă parte, $y\,x$ este o aplicare, deci trebuie să existe E și F astfel încât $y : E \to F$ și $x : E$. Atunci $y\,x : F$.

Exemplu (cont.)

Știm $M = (\lambda zu.\, z)\,(y\,x)$. Am dedus următoarele:

- $x : E$
- $y : E \to F$
- $z : A$ și $z : D$, deci $A \equiv D$
- $u : C$
- $B \equiv C \to D$
- $y\,x : A$ și $y\,x : F$, deci $A \equiv F$.

În concluzie, $A \equiv D \equiv F$, și eliminând redundanțele obținem

$(*)$ $x : E \;\; y : E \to A \;\; z : A \;\; u : C$

Reamintim că aveam $M : B$, adică $M : C \to A$.

Am obținut o schemă generală $(*)$ pentru tipurile lui x, y, z, u care induc un tip pentru M.

Exemplu (cont.)

Știm $M = (\lambda zu.\, z)\,(y\,x)$. Am obținut schema generală

$(*)$ $x : E \;\; y : E \to A \;\; z : A \;\; u : C \;\; M : C \to A$

În schema de mai sus, putem considera tipuri "reale":

- $x : \beta$, $y : \beta \to \alpha$, $z : \alpha$, $u : \delta$, $M : \delta \to \alpha$
- $x : \alpha \to \alpha$, $y : (\alpha \to \alpha) \to \beta$, $z : \beta$, $u : \gamma$, $M : \gamma \to \beta$
  (soluția discutată la Church-typing)
- $x : \alpha$, $y : \alpha \to \alpha \to \beta$, $z : \alpha \to \beta$, $u : \alpha \to \alpha$, $M : (\alpha \to \alpha) \to \alpha \to \beta$

Secțiunea 4

Inferența tipurilor

Inferența tipurilor

Am dori, ca, dat un termen M în λ-calculul fără tipuri, să decidem **algoritmic** dacă există (și, atunci, să și găsim) Γ și τ cu $\Gamma \vdash M : \tau$. Aceasta se numește problema inferenței tipurilor. În continuare, vom prezenta un asemenea algoritm.

Algoritmul va avea nevoie, aproape la fiecare pas, de variabile (de tip) noi, nemaifolosite până atunci (și, deci, în particular, diferite între ele). Vom presupune tacit, în prezentarea sa, că variabilele de tip care apar au această proprietate. (Acesta va fi și punctul sensibil atunci când se încearcă implementarea lui.)

Introducem și noțiunea de **termen adnotat** ca fiind un termen cu tip undeva „între" Church și Curry, în sensul că termenii nu au tipuri, dar, la fiecare λ-abstracțiune, se adaugă și un tip pentru variabila abstractizată. Vom scrie un asemenea termen ca: $\lambda x : \sigma.M$.

Algoritmul

Definim, pentru orice termen M, contextul

$$\Gamma_M := \{x : X \mid x \in FV(M)\}.$$

De asemenea, pentru orice termen M, construim termenul adnotat M', unde toate λx-urile devin $\lambda x : X$.

În continuare, vom defini o funcție c care primește ca argumente: un termen adnotat, un context și o variabilă de tip și care returnează o mulțime de ecuații peste signatura de ordinul I care conține doar simbolul săgeată (vezi discuția de la început).

Finalizare

Definim:

$$c(x, \Gamma \cup \{x : \tau\}, Z) := \{\tau = Z\}$$
$$c(\lambda x : \sigma.M, \Gamma, Z) := c(M, \Gamma \cup \{x : \sigma\}, W) \cup \{Z = \sigma \to W\}$$
$$c(M\ N, \Gamma, Z) := c(M, \Gamma, W_1) \cup c(N, \Gamma, W_2) \cup \{W_1 = W_2 \to Z\}$$

Algoritmul va face apelul $c(M', \Gamma_M, Z)$ și va obține o mulțime de ecuații. Pentru ea, se caută un cgu θ. În caz că nu există, se returnează „eșec”. În caz de succes, rezultatul algoritmului va fi $\widetilde{\theta}(\Gamma_M) \vdash M : \widetilde{\theta}(Z)$ (unde $\widetilde{\theta}$ are semnificația firească).

Putem rula algoritmul pe termenii $(\lambda z.(\lambda u.z))(y\ x)$ și $x\ x$, evidențiați mai devreme.

Secțiunea 5

Limitări ale lambda-calculului cu tipuri simple

Recursie și terminare

Nu mai avem recursie nelimitată deoarece combinatorii de punct fix nu sunt *typeable*.

De exemplu, $\mathbf{Y} \triangleq \lambda y.\,(\lambda x.\,y\,(x\,x))\,(\lambda x.\,y\,(x\,x))$ nu este typeable.

Faptul că orice evaluare se termină este important pentru implementări ale logicilor folosind lambda-calculul.

Tipurile pot fi prea restrictive.

De exemplu, am putea gândi că termenul $(\lambda f.\, \textbf{if}\, (f\, \textbf{T})\, (f\, 3)\, (f\, 5))\, (\lambda x.\, x)$ ar trebui să aibă un tip. Dar nu are!

Soluții posibile:

- Let-polymorphism unde variabilele libere din tipul lui f se redenumesc la fiecare folosire. De exemplu, am putea scrie

  $\textbf{let}\, f = \lambda x.\, x\, \textbf{in}$
  $\textbf{if}\, (f\, \textbf{T})\, (f\, 3)\, (f\, 5)$

- Cuantificatori de tipuri. De exemplu, am avea

  $$\lambda x.\, x : \Pi\alpha\, .\, \alpha \to \alpha$$

  Operatorul de legare Π face explicit faptul că variabila de tip α nu este rigidă.