Izračunljivost u λ-računu

(seminar)

Sandro Lovnički

Sadržaj

0 Uvod

Cilj ovog rada je pokazati kako je λ-račun jedan od mogućih modela za opisivanje izračunljivosti. To ćemo učiniti eksplitinim definiranjem inicijalnih funkcija unutar λ-računa te provođenjem dokaza zatvorenosti λ-definabilnih funkcija na kompoziciju, primitivnu rekurziju i minimizaciju. Kao glavni rezultat toga, slijedit će Kleenejev teorem o ekvivalenciji parcijalno rekurzivnih funkcija te λ-definabilnih funkcija.
Nadalje, u konkretnoj implementaciji interpretera za λ-račun, pokazat ćemo i kako zaista djeluju apstraktni λ-izrazi koji stvaraju primitivne rekurzije i minimizacije. Kao primjere programskih kodova dat ćemo izgradnje nekih klasičnih primjera rekurzivnih funkcija.

0.1 Osnovne definicije λ-računa

U ovom uvodnom potpoglavlju, dat ćemo kratak podsjetnik na osnovne definicije λ-računa, ali rad podrazumijeva da je čitatelj upoznat s λ-računom te će većina stvari biti korištena u skladu s tim. Često će u dokazima biti korišteni λ-izrazi koji možda nisu prije toga eksplicitno definirani, ali čija uloga je jasna.

Definicija 0.1.1 *Lambda izrazi su riječi nad abecedom koju čine:*

- *varijable:* v_0,v_1,...
- *apstraktor:* λ
- *zagrade i točka:* (,),.

Definicija 0.1.2 *Skup λ-izraza Λ definiramo induktivno:*

- $x \in \Lambda$, *gdje je* x *varijabla*
- $M \in \Lambda \Rightarrow \lambda x.M \in \Lambda$
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$

Napomena 0.1.3 *Navedimo ovdje i neke notacijske napomene koje će nam olakšati pisanje i razumijevanje:*

1. *varijable ćemo označavati malim slovima (x,y,z,...), a λ-izraze velikim (M,N,...)*
2. *vanjske zagrade nećemo pisati*

3. $\overrightarrow{x} \equiv x_1, ..., x_n,\ \overrightarrow{N} \equiv N_1, ..., N_n$

4. $\lambda\overrightarrow{x}.M \equiv \lambda x_1.\lambda x_2....\lambda x_n.M$

5. $M\overrightarrow{N} \equiv (...((MN_1)N_2)...N_n)$ *(asocijativnost ulijevo)*

Definicija 0.1.4 *(Supstitucija) Kažemo da je N supstituiran za x u M i pišemo $M[x := N]$ ako smo sve odgovarajuće pojave varijable x u izrazu M zamijenili izrazom N.*

Definicija 0.1.5 *Teorija λ ima formule oblika $M = N$ (M,$N \in \Lambda$) generirane sljedećim aksiomima i pravilima:*

1. $(\lambda x.M)N = M[x := N]$ *(β-konverzija)*

2. $M = M$

3. $M = N \Rightarrow N = M$

4. $M = N, N = L \Rightarrow M = L$

5. $M = N \Rightarrow MZ = NZ$

6. $M = N \Rightarrow ZM = ZN$

7. $M = N \Rightarrow \lambda x.M = \lambda x.N$

8. $\lambda x.Mx = M$ *(η-konverzija)*

1 Rekurzivnost i λ-definabilnost

Bez obzira koji model izračunavanja koristimo (RAM-programe, Turingov stroj, λ-račun, Postove strojeve, ...), definabilnost se bavi definiranjem funkcija unutar danog modela što za cilj ima pokazati koliko je taj model robustan.

Glavna točka ovog poglavlja je pokazati ekvivalenciju skupa funkcija koje je moguće definirati unutar λ-računa i skupa parcijalno rekurzivnih funkcija. Sjetimo se, skup parcijalno rekurzivnih funkcija sastoji se od svih inicijalnih funkcija te je zatvoren na kompoziciju, primitivnu rekurziju i minimizaciju.

1.1 Churchovi numerali

Kako skup Λ eksplicitno ne sadrži nikakve numerale (reprezentacije apstraknog pojma broja), htjeli bismo odabrati λ-izraze koji će ih reprezentirati. Nadalje, želimo definirati i brojevne funkcije nad takvim numeralima. Koristiti ćemo najpoznatiju verziju numerala unutar λ-računa - Churchove numerale.

Definicija 1.1.1 *Definiramo $c_n = c_0, c_1, ...,$ gdje je*

$$c_n = \lambda fx.f^n x = \lambda fx.f(f...(fx)...)$$

Dakle, broj n reprezentiramo numeralom c_n koji prima f i x te vraća vrijednost izraza dobivenog n puta primjenom f na x.

Jasno, nisu sve numeričke funkcije definirane na cijeloj svojoj domeni te izračunavanje takvih nikad ne stane. Kako bismo i tu opciju imali pokrivenu λ-izrazima, uvedimo sljedeću definiciju.

Definicija 1.1.2 *Kažemo da je $M \in \Lambda$ rješiv ako $\exists \overrightarrow{N}$ takvi da je $M\overrightarrow{N} = I$. Ako M nije rješiv, kažemo da je nerješiv.*

Lema 1.1.3 *Svi Churchovi numerali su rješivi λ-izrazi i to za $\overrightarrow{N}$ (iz definicije) $= II$.*

Dokaz. Indukcijom.

- $c_0 II = (\lambda fx.x)II = (\lambda x.x)I = I$
- $c_{n+1}II = (Sc_n)II = ((\lambda nfx.f(nfx))c_n)II = (\lambda fx.f(c_n fx))II =$
  $= (\lambda x.I(c_n Ix))I = I(c_n II) = (pretp.) = II = I$

1.2 Y kombinator

Implementacija rekurzije (koja će nam trebati u sljedećem potpoglavlju) samopozivanjem nije dopuštenja u λ-računu stoga moramo posegnuti za najznačajnijom idejom u teoriji programskih jezika. Konstruirat ćemo λ-izraz Y koji će, primijenjen na nerekurzivnu funkciju f, kreirati njenu "rekurzivnu verziju" tražeći joj fiksnu točku. Y kombinator zovemo i kobinator fiksne točke.

Definicija 1.2.1 $Y := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$.

Propozicija 1.2.2 *Y kombinator je kombinator fiksne točke, tj.*

$$YF = F(YF),\ \forall F \in \Lambda$$

Dokaz. Neka je $W := \lambda x.F(xx)$. Tada je

$$YF = WW = (\lambda x.F(xx))W = F(WW) = F(YF)$$

Napomena 1.2.3 *Kombinatora fiksne točke ima prebrojivo mnogo.*

Uzmimo za primjer funkciju $fakt$ za koju želimo da prima numeral c_n i rekurzivno računa te vraća $c_{n!}$. Intuitivna definicija funkcija faktorijela dana je s

$$fakt := \lambda n.\ if\ (eq\ n\ c_0)\ c_1\ (mul\ n\ (fakt\ (pred\ n)))$$

Kao što smo rekli, samopozivanje nije dopuštano pa apstrahirajmo $fakt$ u tijelu funkcije.

$$fakt' := \lambda f n.\ if\ (eq\ n\ c_0)\ c_1\ (mul\ n\ (f\ (pred\ n)))$$

Ovakva definicija sad je dopustima u λ-računu, ali izgubili smo svojstvo rekurzivnosti.

Lema 1.2.4 *Funkcija $faktorijel := Y\ fakt'$ rekurzivno računa faktorijel, tj. $faktorijel\ c_n = c_{n!}$*

Dokaz. Primijetimo da je definicija dopustiva u λ-računu jer ne sadrži eksplicitno samopozivanje.
Dokaz provodimo indukcijom. Za lakše čitanje dokaza, označimo $yf := \lambda x.fakt'(xx)$. Primijetimo da je tada $Y\ fakt' = yf\ yf = fakt'\ (yf\ yf)$.

- c_0.

$$faktorijel\ c_0 = (Y\ fakt')\ c_0 = fakt'\ (yf\ yf)\ c_0 =$$
$$= if\ (eq\ c_0\ c_0)\ c_1\ (mul\ c_0\ ((yf\ yf)\ (pred\ c_0))) = c_1$$

- Neka tvrdnja vrijedi za neki c_n, tj. $faktorijel\ c_n = c_{n+1}$. Tada imamo (uz početak sličan kao gore)

$$faktorijel\ c_{n+1}\ = ... =$$
$$= if\ (eq\ c_{n+1}\ c_0)\ c_1\ (mul\ c_{n+1}\ ((yf\ yf)\ (pred\ c_{n+1}))) =$$
$$= mul\ c_{n+1}\ (fakt'\ (yf\ yf)\ (pred\ c_{n+1})) =$$
$$= mul\ c_{n+1}\ (faktorijel\ c_n) = (pretp.) = mul\ c_{n+1}\ c_{n!} = c_{(n+1)!}$$

Iz dokaza koraka indukcije vidimo da $faktorijel$ zaista računa na rekurzivan način.

1.3 λ-definabilnost

λ-račun predstavlja određenu klasu funkcija nad prirodnim brojevima. Pokazat će se, po rezultatu Kleeneja, da su to upravo parcijalno rekurzivne funkcije. Krenimo sad s dokazima koji će voditi tom rezultatu.

Definicija 1.3.1 *Neka je φ parcijalna numerička funkcija s p argumenata. Kažemo da je φ λ-definabilna ako postoji $F \in \Lambda$ takav da $\forall \overrightarrow{n} \in \mathbb{N}$*

$$\begin{cases} F\overrightarrow{c_n} = c_{\varphi(\overrightarrow{n})}, & \varphi(\overrightarrow{n}) \text{ je definirana u } \overrightarrow{n} \\ F\overrightarrow{c_n} \text{ je nerješiv}, & \text{inače} \end{cases}$$

Eksplicitno kažemo i da je φ λ-definirana s F.

Lema 1.3.2 *Incijalne funkcije su λ-definabilne.*

Dokaz. Definirajmo odgovarajuće λ-izraze za nul-funkciju, projekciju i sljedbenika.

- $Z := \lambda x.c_0$
  Trivijalno.

- $I_n^k := \lambda x_1, ..., x_k.x_n$
  Trivijalno.

- Neka je f λ-izraz koji predstavlja funkciju sljedbenika i neka x predstavlja nulu. Tada bi, po definiciji Churchovih numerala, $c_n f x$ bio broj n, a njegov sljedbenik $n+1$ bi tada bio $f(c_n f x)$. Dakle, $c_{n+1} f x = f(c_n f x)$. Ako sad apstrahiramo f i x na obje strane, imamo $\lambda f x.c_{n+1} f x = \lambda f x.f(c_n f x)$. Na lijevoj strani iskoristimo aksiom η-konverzije da bismo dobili $c_{n+1} = \lambda f x.f(c_n f x)$ te sad apstrahiramo c_n da bismo dobili funkciju sljedbenika:
$$S := \lambda n f x.f(n f x)$$
Zaista, $S c_n = (\lambda n f x.f(n f x)) c_n = \lambda f x.f(c_n f x) =$
$= \lambda f x.f(\lambda f x.f(f...(f x)...)) f x) = \lambda f x.f(f(f...(f x)...))) = c_{n+1}$.

Kad govorimo o λ-definabilnosti kompozicije parcijalnih funkcija, valja biti oprezan. Naime, reprezentacija kompozicije nije uvijek jednaka kompoziciji reprezentacija funkcija članica što možemo vidjeti kroz sljedeći primjer.

Primjer 1.3.3 *Neka je $\forall n\ f(n) = 0$ i $g(n)$ nedefinirana. Tada je njihova kopozicija $f \circ g$ nedefinirana za svaki n. Ove funkcije možemo definirati λ-izrazima $F = \lambda x.c_0$ i $G = \lambda x.\Omega$, ali tada imamo*

$$FG = (\lambda x.c_0)G = c_0$$

što svakako nije nerješiv izraz.

Lema 1.3.4 *λ-definabilne parcijalne funkcije zatvorene su na kompoziciju.*

Dokaz. Neka su $\chi, \psi_1, ..., \psi_m$ λ-definirane s $G, H_1, ..., H_m$, respektivno.
Tada je
$$\varphi(\overrightarrow{n}) \simeq \chi(\psi_1(\overrightarrow{n}), ..., \psi_m(\overrightarrow{n}))$$
λ-definirana s

$$F \equiv \lambda \overrightarrow{x}.(H_1 \overrightarrow{x} II)...(H_m \overrightarrow{x} II)(G(H_1 \overrightarrow{x})...(H_m \overrightarrow{x}))$$

Lako je za vidjeti da takav F zaista zadovoljava definiciju.

Lema 1.3.5 *λ-definabilne funkcije zatvorene su na primitivnu rekurziju.*

Dokaz. Neka su χ i ψ λ-definirane s G i H, respektivno.
Tada je φ definirana s
$$\varphi(\overrightarrow{n}, 0) = \chi(\overrightarrow{n})$$
$$\varphi(\overrightarrow{n}, k+1) = \psi(\overrightarrow{n}, k, \varphi(\overrightarrow{n}, k))$$

λ-definirana s

$$F \equiv \lambda\overrightarrow{x}.\ Y\ (\lambda f\,\overrightarrow{x}k.\ (isZ\ k)\ (G\overrightarrow{x})\ (H\ \overrightarrow{x}\ (pred\ k)\ (f\ \overrightarrow{x}\ (pred\ k))))$$

Indukcijom po k, vidi se da je $F\overrightarrow{c_n}k = c_{\varphi(\overrightarrow{n},k)}$.

Lema 1.3.6 *λ-definabilne funkcije zatvorene su na minimizaciju.*

Dokaz. Neka je je χ totalna funkcija λ-definirana s G i

$$\varphi(\overrightarrow{n}) \simeq \mu m[\chi(\overrightarrow{n}, m) = 0]$$

Kao pomoćnu funkciju, za neki $P \in \Lambda$ za koji vrijedi ili $Pc_n = T$ ili $Pc_n = F$ za sve n, definirajmo $H_p = Y(\lambda hz.\ Pz\ z\ (h\ (S\ z)))$. Sad minimizaciju izraza P možemo zapisati kao slanje početne vrijednosti $z = 0$ rekurzivnom evaluatoru H_p, tj. $\mu P = H_p c_0$.
Analogno, (za $P = \lambda y.\ isZ\ (G\overrightarrow{x}y)$) izraz definiran s

$$F \equiv \lambda\overrightarrow{x}.\ \mu[\lambda y.\ isZ\ (G\overrightarrow{x}y)]$$

λ-definira φ.

Propozicija 1.3.7 *Parcijalno rekurzivne funkcije su λ-definabilne.*

Dokaz. Slijedi iz dokaza prethodnih lema.

Teorem 1.3.8 *(Kleene) Parcijalna funkcija je parcijalno rekurzivna ako i samo ako je λ-definabilna.*

2 pLam

Kako bismo konstruirane ideje iz prethodnih poglavlja testirali, poslužit ćemo se interpreterom za λ-račun napisanom u Haskellu. Izračunavanje rekurzivnih funkcija vidjet ćemo na primjerima programa napisanih u jeziku čistog λ-računa, koji intepreter pLam prepoznaje.

Kompletan izvorni kod ovog interpretera, kao i upute za njegovo korištenje, mogu se naći na adresi https://github.com/sandrolovnicki/lambda-calculus-interpreter.

2.1 Biblioteka *comp*

Jasno je da konstrukcija iole kompliciranijih izračunavanja zahtijeva postojanje predefiniranih bazičnih i raznih pomoćnih funkcija kako ne bismo svaki puta morali pisati mnogo redaka λ-izraza, već samo nekoliko ključnih. To će nam biti omogućeno naredbom : *import libname*, gdje su u *import/libname.plam* definirani željeni *lambda*-izrazi.

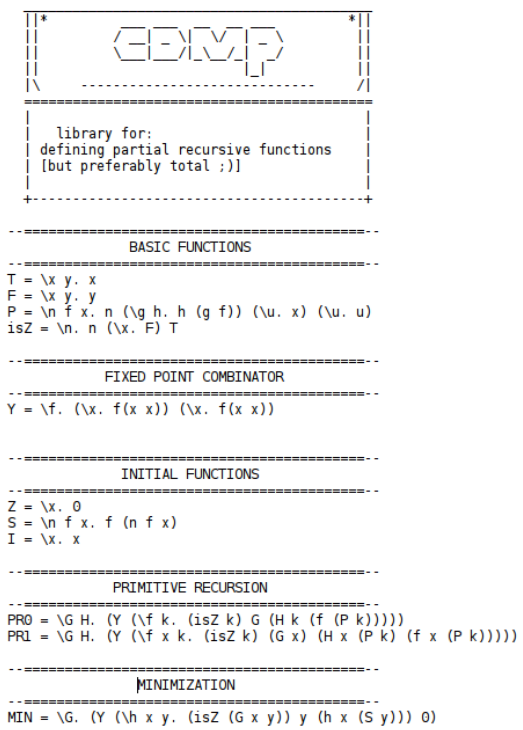

Figure 1: Slika biblioteke *comp*

2.2 Primjeri

```
      _
     | |
  ___| |   ___  __  __
 | _ \ |__| _ \|  \/  |
 |  _/____|___\_\__/_| v0.1.0
 |_| pure lambda-calculus interpreter
=======================================
```

```
pLam> -- defining factorial as primitive recursion of 1 and mul
pLam>
pLam> :import std
pLam> :import comp
pLam>
pLam> factorial = PR0 1 mul
pLam>
pLam> factorial 3
------ result: 6
pLam>
```

```
      _
     | |
  ___| |   ___  __  __
 | _ \ |__| _ \|  \/  |
 |  _/____|___\_\__/_| v0.1.0
 |_| pure lambda-calculus interpreter
=======================================
```

```
pLam> -- minimizing substitution
pLam>
pLam> :import std
pLam> :import comp
pLam>
pLam> testM = MIN sub
pLam>
pLam> testM 3
------ result: 3
pLam>
```

References

[1] Hendrik Pieter Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Elsevier Science B.V., Amsterdam, The Netherlands 1984.

[2] Mayer Goldberg. *The Lambda Calculus*
`http://www.little-lisper.org/website/files/lambda-calculus-tutorial.pdf`