UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Aleš Bizjak

Lambda račun in izračunljive funkcije

Delo diplomskega seminarja

Mentor: izred. prof. dr. Andrej Bauer

Ljubljana, 2010

Kazalo

Povzetek

Lambda račun je abstraktna teorija funkcij. S funkcijami lahko v osnovi počnemo dve stvari. Prva je uporaba funkcije na argumentu, kar imenujemo aplikacija, druga je abstrakcija in ustreza gradnji novih funkcij iz spremenljivk in obstoječih funkcij. Izračunljive funkcije so formalizacija intuitivnega pojma algoritma. Predlaganih je bilo več formalizacij pojma izračunljivosti, a so se vse izkazale za ekvivalentne, mi pa za definicijo vzamemo rekurzivne funkcije Gödla in Herbranda.

V delu najprej defininiramo množico lambda izrazov in relacije na njej ter pokažemo nekaj osnovnih lastnosti teh relacij. Potem definiramo rekurzivne funkcije, ter pokažemo, da lahko vse totalne rekurzivne funkcije izrazimo z ustreznimi lambda izrazi. V zadnjem delu s predstavitvijo nekaj osnovnih podatkovnih struktur in algoritmov pokažemo, da lahko na lambda račun gledamo kot na minimalističen programski jezik.

Abstract

Lambda calculus is an abstract theory of functions. There are two basic operations on functions. The first one is application of a function to an argument, and the second one is building new functions from variables and existing functions, which is called abstraction. Computable functions are a formalization of the intuitive notion of algorithm. There are many different but equivalent formalizations. We take recursive functions of Gödel in Herbrand as the definition.

First we define the set of lambda terms and show some basic properties. Then we define recursive functions and proceed to show that all total recursive functions are lambda definable. In the last part we show that lambda calculus can be viewed as a minimalistic programming language by encoding some data structures and algorithms with lambda terms.

Math. Subj. Class. (2010): 03D20, 03B40
Ključne besede: lambda račun, izračunljive funkcije, rekurzivne funkcije
Keywords: lambda calculus, computable functions, recursive functions

1. Uvod

Funkciji pravimo, da je *dejansko izračunljiva*, če obstaja algoritem za njen izračun; torej če obstaja končen spisek navodil, ki v principu zadošča, da bi računalnik izračunal vrednost funkcije za podane argumente. Jasno je, da pojem dejanske izračunljivosti ni formalen, ampak intuitiven in to je težava, sploh če hočemo pokazati da neka funkcija ni izračunljiva, zato je bil v začetku 20. stoletja interes formalizirati pojem dejanske izračunljivosti, saj so začeli nastajati vse bolj zmogljivi računski stroji in z njimi povezana vprašanja.

V 30-ih letih prejšnjega stoletja je bilo tako predlaganih več formalizmov za definicijo dejanske izračunljivosti. Prva dva, in v začetku najpomembnejša, sta bila Churchev lambda račun in splošne rekurzivne funkcije Gödla in Herbranda. Kmalu so uspeli pokazati, da sta formalizma ekvivalentna, torej da se v obeh da izraziti isti razred funkcij, in zato je kmalu Church predlagal, da bi vzeli splošne rekurzivne funkcije za definicijo razreda dejansko izračunljivih funkcij. Ta predlog je danes znan kot Churcheva teza.

Kmalu za tem je Alan Turing objavil članek [Tur37], v katerem je definiral Turingove stroje in v dodatku tudi pokazal, da njegovi stroji izračunajo enak razred funkcij kot lambda račun. Turingova analiza, kako bi deloval človek, "računalnik", ki bi mehansko sledil navodilom za izračun, ki je osnova za Turingove stroje, skupaj z dokazi, da so vsi trije formalizmi ekvivalentni, je bila močan argument, da je Churcheva teza smiselna.

Načrt dela je naslednji: Najprej bomo definirali lambda račun in pokazali dovolj lastnosti, da bomo znali dokazati, da so izračunljive funkcije izrazljive v lambda računu. Za definicijo izračunljivih funkcij bomo vzeli definicijo iz Churcheve teze, torej splošne rekurzivne funkcije.

2. Lambda račun

Lambda račun je abstraktna teorija funkcij in je bil zasnovan z namenom, da bi služil kot osnova konstruktivne matematike. Kmalu sta Churcheva študenta, Kleene in Rosser pokazala, da za ta namen ni dober, kljub temu pa je del prvotnega lambda računa, ki je tema tega dela, postal uporaben v študiju izračunljivosti in kasneje v študiju semantike programskih jezikov v računalništvu. V 60-ih letih so opazili povezavo med procedurami (z nekaj omejitvami) v programskem jeziku ALGOL in lambda izrazi [Lan65], še bolj neposreden vpliv pa je imel lambda račun na programski jezik LISP in kasneje na ostale funkcijske programske jezike.

Oglejmo si motivacijo za kasnejšo definicijo lambda izrazov. S funkcijami lahko v osnovi počnemo dve stvari. Lahko jih apliciramo na argumente, kar, jasno, imenujemo aplikacija, ali pa iz že obstoječih funkcij in spremenljivk gradimo nove funkcije, kar imenujemo abstrakcija.

V običajnem matematičnem zapisu pišemo aplikacijo funkcije f na argument a kot $f(a)$ ali $f\,a$. V lambda računu izberemo slednji zapis.

Abstrakcijo običajno pišemo kot $f(x) = M$, ali, če funkciji ne damo imena, kot $x \mapsto M$, kjer je M nek izraz, v katerem x lahko prosto nastopa. V lambda računu

isto funkcijo zapišemo kot $\lambda x.M$.[1] Formalizacija aplikacije in abstrakcije, skupaj z nekaj naravnimi pravili, dá lambda račun.

Tukaj je na mestu opomba. Ločiti moramo dva pogleda, sintaktičnega in semantičnega. V nadaljevanju predstavljen lambda račun bo sintaktična teorija, torej definirali bomo neko množico izrazov in relacij na izrazih. Vmes si bomo za motivacijo definicij in lažje razumevanje predstavljali lambda izraze kot funkcije, vendar moramo biti pri tem pazljivi, saj ni enostavno zgraditi modela lambda računa, torej interpretirati lambda izraze kot funkcije med nekimi množicami. Več o tem bomo povedali v dodatku B.

2.1. **Definicija in osnovne lastnosti.** Za začetek moramo uvesti nekaj novih pojmov.

Definicija 2.1. *Lambda izrazi so besede nad abecedo, sestavljeno iz:*

- *neskončne množice spremenljivk* $v_0, v_1, \ldots$,
- *abstraktorja* λ,
- *predklepaja in zaklepaja* (,).

Množico lambda izrazov Λ *definiramo induktivno:*

(1) $x \in \Lambda$,
(2) $M \in \Lambda \wedge N \in \Lambda \Rightarrow (MN) \in \Lambda$,
(3) $M \in \Lambda \Rightarrow (\lambda x.M) \in \Lambda$,

kjer je x *v* (1) *in* (3) *poljubna spremenljivka. Izraze pod točko* (2) *imenujemo aplikacija, izraze pod točko* (3) *pa abstrakcija.*

Primeri 2.2. *Naslednji izrazi so lambda izrazi:*

(1) v_0
(2) $(v_0\, v_1)$
(3) $(\lambda v_0.((\lambda v_1.v_1)\, v_0))$

Da bodo izrazi bolj pregledni, bomo privzeli nekaj okrajšav v njihovi pisavi:

- če ne bo dvoumnosti, bodo velike tiskane črke označevale poljubne lambda izraze, male tiskane črke $x, y, z, \ldots$ bodo predstavljale poljubne spremenljivke in različni črki bosta, če ne bo drugače poudarjeno, predstavljali dve različni spremenljivki,
- za $n \geq 1$ bo $M_1 M_2 \cdots M_n$ pomenilo $(\cdots (M_1 M_2) M_3) \cdots M_n)$, torej aplikacija je levo asociativna,
- za $n \geq 1$ bo $\lambda x_1 x_2 \ldots x_n.M$ pomenilo $(\lambda x_1.(\lambda x_2.(\cdots (\lambda x_n.M) \cdots)))$,
- $\lambda x.MN$ bo pomenilo $(\lambda x.(MN))$.

Zavedati se moramo, da je to samo dogovor, ki olajša pisavo in branje, še zmeraj pa imajo lambda izrazi formalno enako sintakso, kot smo jo definirali v 2.1.

Sintaktično enakost dveh izrazov bomo označili z $\equiv$.

Z uporabo tega dogovora lahko sedaj bolj pregledno zapišemo nekaj lambda izrazov.

Primeri 2.3. *Naslednji izrazi predstavljajo lambda izraze:*

- $\lambda x.x\, x$,

[1] Zakaj λ? Razlog ni znan. Church je podal v razmaku nekaj let dva nasprotujoča si razloga. Prvi je bil, da je bila izbira črke λ motivirana z notacijo, ki sta jo uporabljala Russell in Whitehead za funkcijsko abstrakcijo v delu Principia Mathematica. Drugi podan razlog je bil, da je enostavno potreboval neko oznako in je izbral črko λ brez posebnih razlogov.

- $\lambda y.(\lambda x.y\,(x\,x))\,(\lambda x.y\,(x\,x))$,
- $\lambda x\,y.x$,
- $\lambda x\,y.y$,
- $\lambda x\,y\,z.y\,(x\,y\,z)$.

Definicija 2.4. *Množico podizrazov izraza M, $sub(M)$, definiramo rekurzivno:*

- $sub(x) = \{x\}$,
- $sub(\lambda x.M) = sub(M) \cup \{\lambda x.M\}$,
- $sub(M\,N) = sub(M) \cup sub(N) \cup \{M\,N\}$.

Izraz N je podizraz izraza M, če je $N \in sub(M)$.

Izraz M lahko nastopa v izrazu N večkrat. Npr. izraz x nastopa v izrazu $x\,y\,z\,(\lambda x.x\,x)$ na treh mestih. Nastop izraza M v izrazu N lahko definiramo formalno kot par (M, i), kjer je i nek indikator položaja, na katerem nastopa M v N. Formalizacije ne bomo potrebovali, zato je dovolj samo predstava in zavedanje, da se da narediti.

Definicija 2.5. *Množico prostih spremenljivk izraza M, $FV(M)$, definiramo rekurzivno:*

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

Spremenljivka x nastopa prosto *v izrazu M, če je $x \in FV(M)$. Sicer nastopa* vezano *ali pa sploh ne nastopa.*

V izrazu $\lambda x.M$ imenujemo izraz M doseg *λx.*

Izraz, v katerem proste spremenljivke ne nastopajo, je zaprt *izraz.*

Alternativno bi lahko definirali, da je spremenljivka x vezana, če je v dosegu λx, sicer pa je prosta. Spremenljivka lahko v izrazu nastopa na več mestih in zato je lahko na nekaterih mestih njen nastop vezan, na drugih pa prost.

Primer 2.6. *Naj bo $M \equiv (\lambda x.x\,y)\,u\,(\lambda z.z)$. Potem je $FV(M) = \{y, u\}$ in $sub(M) = \{x, y, x\,y, u, z, \lambda x.x\,y, \lambda z.z, (\lambda x.x\,y)\,u, M\}$.*

Osnovna pretvorba, ki jo lahko naredimo med izrazi je, da zamenjamo dele enega izraza z nekim drugim.

2.2. **Substitucija.** S substitucijo zamenjamo proste nastope spremenljivke v izrazu z nekim drugim izrazom. Substitucijo prostih nastopov spremenljivke x v izrazu M z izrazom N zapišemo $M\,[x \to N]$. Pri formalni definiciji substitucije moramo biti zelo pozorni, saj lahko drugače hitro naletimo na težave.

Poglejmo npr. izraz $(\lambda x.y)\,[y \to w]$. Izraz $\lambda x.y$ si lahko predstavljamo kot konstantno funkcijo, ki vse argumente slika v y, in če naivno naredimo zapisano substitucijo bi pričakovali, da dobimo konstantno funkcijo, ki vse argumente slika v w. To tudi dobimo, če je $w \not\equiv x$. Če pa naivno naredimo substitucijo $(\lambda x.y)\,[y \to x]$ dobimo izraz $\lambda x.x$, ki si ga lahko predstavljamo kot identiteto, ne pa kot konstantno funkcijo.

Definicija 2.7. *Naj bosta M in N poljubna lambda izraza in x poljubna spremenljivka. $M\,[x \to N]$ definiramo rekurzivno:*

(1) $x\,[x \to N] \equiv N$
(2) $x\,[y \to N] \equiv x$ *za vsak* $y \not\equiv x$
(3) $(PQ)\,[x \to N] \equiv (P\,[x \to N]\;Q\,[x \to N])$
(4) $(\lambda x.P)\,[x \to N] \equiv \lambda x.P$

(5) $(\lambda y.P)\,[x \to N] \equiv \lambda y.P$, *če* x *ne nastopa prosto v* P.
(6) $(\lambda y.P)\,[x \to N] \equiv \lambda y.(P\,[x \to N])$, *če* x *nastopa prosto v* P *in* y *ne nastopa prosto v* N.
(7) $(\lambda y.P)\,[x \to N] \equiv \lambda z.(P\,[y \to z]\,[x \to N])$, *če* x *nastopa prosto v* P *in* y *nastopa prosto v* N *in je* z *prva spremenljivka, ki ne nastopa niti v* P *niti v* N. *Taka zagotovo obstaja, saj je množica spremenljivk neskončna, izrazi pa so vedno končni.*

Substitucijo $(\lambda x.y)\,[y \to x]$ lahko sedaj naredimo, z uporabo točke (7), kot

$$(\lambda x.y)\,[y \to x] \equiv \lambda z.(y\,[x \to z]\,[y \to x]) \equiv \lambda z.(y\,[y \to x]) \equiv \lambda z.x,$$

kar je pričakovani rezultat; konstantna funkcija, ki vsak argument slika v x.

Sedaj bomo rigorozno pokazali nekaj lastnosti substitucije, ki jih bomo kasneje potrebovali. Dokazi bodo večinoma z indukcijo na strukturo izraza ali pa z indukcijo na dolžino izraza. Lambda izraz si lahko predstavljamo kot drevo, ki ima v listih spremenljivke, v notranjih vozliščih pa iz poddreves sestavimo z aplikacijo in abstrakcijo nove izraze. Dokaz z indukcijo na strukturo izraza poteka tako, da lastnost dokažemo za spremenljivke, potem pa pokažemo, da se ohranja pri aplikaciji in abstrakciji. Tako potem pridemo iz listov do korena drevesa, podobno kot pri navadni indukciji pridemo od baze indukcije do želenega naravnega števila.

Nekaj dokazov bo lažjih, če jih bomo delali z indukcijo na dolžino izraza.

Definicija 2.8. *Dolžino lambda izraza definiramo rekurzivno s predpisi:*
(1) $lgh(x) = 1$ *za vse spremenljivke* x,
(2) $\forall M, N \in \Lambda : lgh(M\,N) = lgh(M) + lgh(N)$,
(3) $\forall M \in \Lambda : lgh(\lambda x.M) = lgh(M) + 1$.

Lema 2.9. *Za lambda izraz* M *in spremenljivki* x *in* y *velja:* $lgh(M) = lgh(M\,[x \to y])$.

Dokaz je preprost z indukcijo na dolžino izraza M po točkah definicije 2.7.

Lema 2.10. *Če* x *ne nastopa prosto v* P *velja* $P\,[x \to N] \equiv P$.

Dokaz. Z indukcijo na strukturo izraza. Za spremenljivke lema velja po točki (2) definicije substitucije.

Naj bo $P \equiv P_1P_2$ in predpostavimo, da lema že velja za P_1 in P_2. Če x ne nastopa prosto izrazu P, ne nastopa prosto niti v izrazih P_1 in P_2 in je, po točki (3) definicije substitucije, $P\,[x \to N] \equiv (P_1\,[x \to N]\ P_2\,[x \to N])$, kar je po indukcijskih predpostavkah za P_1 in P_2 enako $P_1P_2 \equiv P$.

Naj bo $P \equiv \lambda x.M$ kjer je M nek lambda izraz. Potem je, po točki (4) definicije substitucije, $(\lambda x.P)\,[x \to N] \equiv P$.

Naj bo $P \equiv \lambda y.M$. Po predpostavki x v M ne nastopa prosto, zato je po točki (5) definicije substitucije $(\lambda y.M)\,[x \to N] \equiv \lambda y.M \equiv P$. □

Posledica 2.11. *Naj bo* $n \in \mathbb{N}$. *Če nobena izmed* $x_1, x_2, \ldots, x_n$ *ne nastopa prosto v* P, *je*

$$P\,[x_1 \to M_1]\,[x_2 \to M_2] \cdots [x_n \to M_n] \equiv P.$$

Dokaz. Z indukcijo na število substitucij ob uporabi prejšnje leme. □

Posledica 2.12. *Naj bosta* $i, n \in \mathbb{N}$ *in naj dodatno velja* $1 \leq i \leq n$. *Če nobena izmed* $x_1, x_2, \ldots, x_n$ *ni prosta v nobenem izmed* $M_1, M_2, \ldots, M_n$, *je*

$$x_i\,[x_1 \to M_1] \cdots [x_n \to M_n] \equiv M_i.$$

Dokaz. Z indukcijo na število substitucij. Če je $n = 1$ mora biti tudi $i = 1$ in je zato $x_1 [x_1 \to M]$ po točki (1) definicije substitucije enako M_1.

Privzemimo zdaj, da trditev velja za n substitucij. Imamo dve možnosti:

$i = n + 1$**:** po posledici 2.11 je

$$x_{n+1} [x_1 \to M_1] \cdots [x_n \to M_n] \equiv x_{n+1}$$

in zato

$$x_{n+1} [x_1 \to M_1] \cdots [x_n \to M_n] [x_{n+1} \to M_{n+1}] \equiv M_{n+1}$$

po točki (1) definicije substitucije.

$i < n + 1$**:** po indukcijski predpostavki je

$$x_i [x_1 \to M_1] \cdots [x_n \to M_n] \equiv M_i$$

in zato po lemi 2.10, uporabljeni na izrazu $M_i [x_{n+1} \to M_{n+1}]$,

$$x_i [x_1 \to M_1] \cdots [x_n \to M_n] [x_{n+1} \to M_{n+1}] \equiv M_i.$$

□

Lema 2.13. *Naj bo $n \in \mathbb{N}$. Če nobena izmed $x_1, x_2, \ldots, x_n$ ne nastopa prosto v N in x ni enak nobenemu izmed $x_1, x_2, \ldots, x_n$, velja*

$$(\lambda x_1\, x_2\, \ldots\, x_n.M) [x \to N] \equiv \lambda x_1\, x_2 \ldots\, x_n.(M [x \to N]).$$

Dokaz. Če x ne nastopa prosto v M, ne nastopa prosto v celem izrazu. Uporabimo lemo 2.10 in smo končali.

Oglejmo si torej primer, ko x nastopa prosto v M. Dokazujemo z indukcijo na število spremenljivk $x_1, x_2, \ldots, x_n$. Če je $n = 1$ trditev sledi po točki (6) definicije substitucije.

Privzemimo sedaj, da trditev velja za poljuben nabor n spremenljivk. Potem po točki (6) definicije substitucije velja

$$(\lambda x_1\, x_2\, \ldots\, x_n.M) [x \to N] \equiv \lambda x_1.((\lambda x_2\, x_3 \ldots\, x_n.M) [x \to N]),$$

kar je po indukcijski predpostavki enako

$$\begin{aligned} &\equiv \lambda x_1.(\lambda x_2\, x_3 \ldots\, x_n.M [x \to N]), \\ &\equiv \lambda x_1\, x_2 \ldots\, x_n.M [x \to N]. \end{aligned}$$

□

Lema 2.14. *Naj bosta M in N lambda izraza in x spremenljivka, ki ne nastopa prosto v N. Potem velja*

$$FV(M [x \to N]) \subseteq (FV(M) \cup FV(N)) \setminus \{x\}.$$

Dokaz je enostaven z indukcijo na dolžino izraza M po točkah definicije 2.7.

Posledica 2.15. *Naj bosta $n, m \in \mathbb{N}$. Če nobena izmed $x_1, \ldots, x_n$ ne nastopa prosto v nobenem izmed $N_1, N_2, \ldots, N_n$ in velja $\{y_1, \ldots, y_m\} \cap \{x_1, \ldots, x_n\} = \emptyset$, je izraz*

$$(\lambda x_1\, \ldots\, x_n.M) [y_1 \to N_1] \cdots [y_m \to N_m]$$

sintaktično enak izrazu

$$\lambda x_1\, \ldots\, x_n.(M [y_1 \to N_1] \cdots [y_m \to N_m]).$$

Dokaz. Z indukcijo na število substitucij ob uporabi prejšnjih dveh lem. □

Lema 2.16. *Če x ne nastopa prosto v L in ni nobena spremenljivka vezana v M prosta v N ali L, velja*

$$M\left[x \to N\right]\left[y \to L\right] \equiv M\left[y \to L\right]\left[x \to N\left[y \to L\right]\right],$$

Dokaz. Z indukcijo na strukturo izraza. Predpostavimo, da je spremenljivka z različna od spremenljivk x in y.

M je spremenljivka: Če je $M \equiv x$ je leva stran enaka $N\left[y \to L\right]$ po uporabi točke (1) definicije substitucije, desna stran pa po uporabi točk (2) in (1).

Če je $M \equiv y$ je leva stran enaka L po uporabi točk (2) in (1) definicije substitucije, desna stran pa je enaka L po uporabi točke (1) definicije in leme 2.10.

Če je $M \equiv z$ sta obe strani enaki z po dveh uporabah točke (2) definicije.

$M \equiv PQ$: Enakost sledi iz točke (3) definicije substitucije in indukcijske predpostavke za P in Q.

$M \equiv \lambda x.P$: Leva stran je po točkah (4) in (6) definicije enaka $\lambda x.P\left[y \to L\right]$, desna pa po točkah (6) in (4).

$M \equiv \lambda y.P$: Ker po predpostavki y ni prosta v N uporabimo točko (6) definicije substitucije in zatem še točko (4) ter dobimo na levi strani $\lambda y.P\left[x \to N\right]$. Na desni strani prav tako uporabimo točki (4) in (6) definicije ter lemo 2.10 na substituciji $N\left[y \to L\right]$ in dobimo $\lambda y.P\left[x \to N\right]$.

$M \equiv \lambda z.P$: Po posledici 2.15 je leva stran enaka

$$\left(\lambda z.P\right)\left[x \to N\right]\left[y \to L\right] \equiv \lambda z.P\left[x \to N\right]\left[y \to L\right],$$

kar je po indukcijski predpostavki za P enako

$$\equiv \lambda z.P\left[y \to L\right]\left[x \to N\left[y \to L\right]\right],$$

kar je zopet po posledici 2.15 enako

$$\equiv M\left[y \to L\right]\left[x \to N\left[y \to L\right]\right]. \qquad \square$$

Posledica 2.17. *Če y ne nastopa prosto v P in x ne nastopa prosto v Q in ni nobena spremenljivka vezana v M prosta v P ali Q, velja*

$$M\left[y \to Q\right]\left[x \to P\right] \equiv M\left[x \to P\right]\left[y \to Q\right].$$

Dokaz. Po prejšnji lemi je $M\left[y \to Q\right]\left[x \to P\right] \equiv M\left[x \to P\right]\left[y \to Q\left[x \to P\right]\right]$ in ker x ni prosta v Q, je $Q\left[x \to P\right] \equiv Q$ po lemi 2.10. $\square$

2.3. **Preimenovanje vezanih spremenljivk.** Relacija $\equiv$ je preveč restriktivna, saj $\lambda x.x \not\equiv \lambda y.y$, vendar pa si oba izraza predstavljamo kot isto funkcijo, identiteto. Zato uvedemo novo relacijo enakosti, v kateri so izrazi, ki se razlikujejo samo v poimenovanjih vezanih spremenljivk, enaki.

Definicija 2.18 (sprememba vezanih spremenljivk)**.** *Naj podizraz $\lambda x.M$ nastopa v izrazu N in naj spremenljivka y v izrazu M ne nastopa prosto.* Sprememba vezanih spremenljivk *v izrazu N je zamenjava podizraza $\lambda x.M$ z izrazom $\lambda y.(M\left[x \to y\right])$.*

Spremembo vezanih spremenljivk imenujemo tudi α-pretvorba.

Definicija 2.19 (α-ekvivalenca)**.** *Izraza M in N sta α-enaka oz. enaka z α-pretvorbo oz. kongruentna, če obstaja končno zaporedje (lahko tudi prazno) α-pretvorb, ki pretvori M v N.*

Da sta izraza kongruentna bomo označili z $M \equiv_\alpha N$.

Definicija α-ekvivalence na videz ni simetrična. Izkaže se, da je simetrična in dokaz tega dejstva je relativno dolg, zato je v dodatku A. Težava je, da vsake α-pretvorbe ne moremo obrniti s samo eno α-pretvorbo in se je zato potrebno malo bolj potruditi pri dokazu.

Primeri 2.20. *Nekaj primerov kongruentih, vendar ne sintaktično enakih izrazov in nekongruentnih izrazov.*

- $\lambda x.x\,x \equiv_\alpha \lambda y.y\,y$, *vendar* $\lambda x.x\,x \not\equiv \lambda y.y\,y$,
- $\lambda x.x\,(\lambda y.y\,(\lambda z.z)) \equiv_\alpha \lambda u.u\,(\lambda v.v\,(\lambda w.w))$
- $\lambda x.x\,z \not\equiv_\alpha \lambda y.y\,u$.
- $x \not\equiv_\alpha y$.

Kot se običajno v matematiki ne oziramo na konkretno ime vezane spremenljivke (funkciji $f(x) = x^2$ in $g(y) = y^2$ imamo za enaki), je tudi v lambda računu smiselno obravnavati lambda izraze do relacije α-ekvivalence natančno.

Do sedaj še nismo definirali nobene operacije, s katero bi lahko v lambda računu "računali", npr. da bi lahko izrazili $3 + 2 \mapsto 5$. To bomo sedaj popravili.

2.4. **β-redukcija.** Izraz $(\lambda x.M)\,N$ si lahko predstavljamo kot uporabo funkcije, ki argument x slika v izraz M, v katerem x lahko prosto nastopa, na izrazu N. Rezultat seveda dobimo tako, da vse proste nastope spremenljivke x v M zamenjamo z izrazom N.

Definicija 2.21 (β-redeks)**.** *Vsak izraz oblike*

$$(\lambda x.M)N$$

imenujemo β-redeks.

Definicija 2.22 (β-krčenje)**.** *Naj bosta M in N lambda izraza. β-krčenje definiramo kot dvomestno relacijo, podano z naslednjimi induktivnimi pravili:*

(1) $(\lambda x.M)\,N \mapsto M\,[x \to N]$,
(2) $\forall P \in \Lambda : M \mapsto N \Rightarrow (P\,M) \mapsto (P\,N)$,
(3) $\forall P \in \Lambda : M \mapsto N \Rightarrow (M\,P) \mapsto (N\,P)$,
(4) $\forall x : M \mapsto N \Rightarrow (\lambda x.M) \mapsto (\lambda x.N)$.

Točka (1) definicije neformalno pove ravno to, kar smo hoteli, da funkcijo $\lambda x.M$ lahko izračunamo v točki N tako, da vse proste nastope spremenljivke x zamenjamo z N.

Iz definicije razberemo, da če $M \mapsto N$, potem obstaja β-redeks, podizraz izraza M, recimo $(\lambda x.P)\,Q$, da z zamenjavo tega β-redeksa z izrazom $P\,[x \to Q]$ dobimo izraz N.

Če obstaja končno zaporedje β-krčenj, da velja

$$M \equiv M_1 \equiv_\alpha M_1' \mapsto M_2 \equiv_\alpha M_2' \mapsto \cdots \mapsto M_n \equiv_\alpha N,$$

pišemo $M \rhd_\beta N$ in pravimo, da obstaja β-redukcija iz M v N.

Primeri 2.23. *Nekaj primerov β-krčenj:*

- $(\lambda x.x)\,y \mapsto y$,
- $(\lambda x.y)M \mapsto y$,
- $(\lambda x.x\,x)\,y \mapsto y\,y$,
- $(\lambda x\,y.x\,y\,x\,x\,y)\,(\lambda z.z) \mapsto \lambda y.(\lambda z.z)\,y\,(\lambda z.z)\,(\lambda z.z)\,y \mapsto \lambda y.y\,(\lambda z.z)\,(\lambda z.z)\,y$,
- $(\lambda x.x\,x\,y)\,(\lambda x.x\,x\,y) \mapsto (\lambda x.x\,x\,y)\,(\lambda x.x\,x\,y)\,y \mapsto (\lambda x.x\,x\,y)\,(\lambda x.x\,x\,y)\,y\,y$.

Zadnji primer tudi pokaže, da z β-krčenji ne dobimo vedno enostavnejših izrazov.

Zelo pomembni so izrazi, v katerih ni β-redeksov. Ti intuitivno predstavljajo konec računanja.

Definicija 2.24. *Izraz, v katerem β-redeksi ne nastopajo, je v* β-normalni obliki.
Če velja $M \rhd_\beta N$ in je N v β-normalni obliki, je N β-normalna oblika *izraza M.*

Primeri 2.25. *Naslednji izrazi so v β-normalni obliki:*

- x,
- $\lambda x.x\,x$,
- $\lambda n\,f\,x.n\,(n\,f\,x)$,
- $x\,(\lambda x.x\,x)$.

Obstajajo izrazi, ki nimajo normalne oblike, torej ne obstaja β-redukcija iz izraza do izraza z β-normalno obliko. Npr. izraz $(\lambda x.x\,x)\,(\lambda x.x\,x)$ ni v normalni obliki, saj je β-redeks in β-krčenje nam da izraz $(\lambda x.x\,x)\,(\lambda x.x\,x)$, ki je enak prvotnemu. To je tudi edino možno β-krčenje in zato ta izraz nima β-normalne oblike.

β-redukcija ni simetrična, vendar lahko definiramo simetrično relacijo β-ekvivalence.

Definicija 2.26 (β-enakost)**.** *Izraz M je β-enak izrazu N, pišemo $M =_\beta N$, natanko tedaj, ko obstajajo $M_1, \ldots, M_n$, kjer je $n \in \mathbb{N}_0$, da za vsak i, $0 \leq i < n$ velja ali $M_i \mapsto M_{i+1}$, ali $M_{i+1} \mapsto M_i$ ali $M_i \equiv_\alpha M_{i+1}$, kjer je $M_0 \equiv M$ in $M_n \equiv N$.*

Iz definicije takoj sledi, da je ta relacija ekvivalenčna, porojena z relacijo $\rhd_\beta$. Relacija β-enakosti je zanimiva, ker, intuitivno, dva β-enaka izraza predstavljata isto funkcijo. V dodatku B bomo dokazali, da se dva β-enaka izraza slikata z ustrezno preslikavo v isti element modela lambda računa, kar pokaže, da predstavljata isto funkcijo.

Druga skrajnost, ki nas mora skrbeti, je, da smo mogoče definirali preveliko relacijo, torej tako, da je preveč izrazov β-enakih. Posebej hočemo, da dva lambda izraza M in N, ki sta v normalni obliki, in nista kongruentna, nista β-enaka. Izkaže se, da je relacija β-enakosti ustrezna, vendar pa dokaz tega dejstva ni enostaven. V dodatku B bomo za podmnožico izrazov v β-normalni obliki pokazali, da res niso β-enaki. Splošne lastnosti ne bomo potrebovali. Njen dokaz poteka klasično preko 1. in 2. Church-Rosserjevega izreka. Dokaza sta npr. v [CF58] na straneh 108-144.

Z osnovnimi definicijami smo na tem mestu zaključili. Posvetimo se uporabi.

2.5. **Moč lambde.** Lambda račun je zelo preprost formalizem in iz opisanega se ne zdi, da lahko v lambda računu izrazimo vse izračunljive funkcije, vendar bomo kmalu pokazali, da je tako. Izračunljive funkcije bomo gledali na n-tericah naravnih števil, zato moramo najprej definirati izraze, ki bodo predstavljali naravna števila. Možnih je več predstavitev in vsaka ima svoje prednosti in slabosti. Mi bomo izbrali Scottove numerale.

Definicija 2.27 (Scottovi numerali)**.** *Definiramo jih induktivno:*

- $\overline{0} \equiv \lambda x\,y.x$,
- $\overline{n+1} \equiv \lambda x\,y.y\,\overline{n}$.

Hitro lahko opazimo, da so vsi Scottovi numerali zaprti izrazi in v normalni obliki. Ko tako predstavimo naravna števila, lahko definiramo nekaj uporabnih funkcij. Enomestno konstantno funkcijo, ki vsak argument slika v 0 lahko izrazimo z lambda izrazom

$$\mathbf{Z} \equiv \lambda x.\overline{0}.$$

Enomestno funkcijo naslednik lahko izrazimo z lambda izrazom

$$\mathbf{N} \equiv \lambda n\, x\, y.y\, n.$$

Lema 2.28. *Za vsak $n \in \mathbb{N}_0$ velja $\mathbf{N}\,\overline{n} \rhd_\beta \overline{n+1}$.*

Dokaz. Po definiciji: $\mathbf{N}\,\overline{n} \equiv (\lambda n\, x\, y.y\, n)\,\overline{n} \mapsto \lambda x\, y.y\,\overline{n} \equiv \overline{n+1}$. □

Enomestno funkcijo predhodnik lahko izrazimo z lambda izrazom

$$\mathbf{P} \equiv \lambda n.n\,\overline{0}\,(\lambda x.x).$$

Lema 2.29. *Za vsak $n \in \mathbb{N}$ velja $\mathbf{P}\,\overline{n} \rhd_\beta \overline{n-1}$, $\mathbf{P}\,\overline{0}$ pa je β-enako $\overline{0}$.*

Dokaz. Naj bo najprej $n \in \mathbb{N}$.

$$\mathbf{P}\,\overline{n} \equiv (\lambda n.n\,\overline{0}\,(\lambda x.x))\,\overline{n} \rhd_\beta (\lambda x\, y.y\,\overline{n-1})\,\overline{0}\,(\lambda x.x) \rhd_\beta (\lambda x.x)\,\overline{n-1} \rhd_\beta \overline{n-1}$$

Preverimo še za $\overline{0}$.

$$\mathbf{P}\,\overline{0} \equiv (\lambda n.n\,\overline{0}\,(\lambda x.x))\,\overline{0} \rhd_\beta \overline{0}\,\overline{0}\,(\lambda x.x) \rhd_\beta (\lambda xy.x)\,\overline{0}\,(\lambda x.x) \rhd_\beta \overline{0}$$ □

Iz diskusije po definiciji β-enakosti sledi definicija lambda izrazljivih funkcij.

Definicija 2.30. *Funkcija $f : \mathbb{N}_0^n \to \mathbb{N}_0$ je lambda izrazljiva, če obstaja lambda izraz F, da je*

$$\forall n_1, n_2, \ldots, n_n \in \mathbb{N}_0 : F\,\overline{n_1}\,\overline{n_2} \cdots \overline{n_n} \rhd_\beta \overline{f(n_1, n_2, \ldots, n_n)}.$$

Potem rečemo, da F lambda izraža f. Lambda izraz, ki izraža funkcijo ψ, bomo pisali kot $\overline{\psi}$.

3. Izračunljive funkcije

V tridesetih letih 20. stoletja so neodvisno nastali tri koncepti izračunljivosti. Funkcije izračunljive s Turingovimi stroji, lambda izrazljive funkcije in splošne rekurzivne funkcije. Vsak formalizem ima svoje prednosti in slabosti. Mi bomo za definicijo izračunljivih funkcij vzeli splošne rekurzivne funkcije. Razlog leži v relativno kratkem dokazu, da so splošne rekurzivne funkcije lambda izrazljive in pa v tem, da je bila Churcheva teza najprej izražena s splošnimi rekurzivnimi funkcijami. Dokaz, da je razred lambda izrazljivih funkcij enak razredu funkcij, ki jih izračunajo Turingovi stroji, bi presegel okvirje tega seminarja.

Pri študiju izračunljivosti funkcij se bomo omejili na numerične funkcije, to so funkcije $f : D \subseteq \mathbb{N}_0^k \to \mathbb{N}_0$. Funkcije $f : D \subseteq \mathbb{N}_0^k \to \mathbb{N}_0$ bomo imenovali *delne* ali *parcialne*, njihovo podmnožico, funkcije $f : \mathbb{N}_0^k \to \mathbb{N}_0$, pa *totalne*. Za totalne funkcije bomo rigorozno pokazali, da so lambda izrazljive, za delne pa bomo samo namignili kje so težave in kako jih premagati. Celoten dokaz bi zahteval še kar nekaj podporne teorije.

Ker bomo delali s funkcijami več spremenljivk in pisanje posameznih spremenljivk postane dolgo in zamudno, bomo uporabili vektorske oznake. Funkcijo f uporabljeno na argumentih $(x_1, x_2, \ldots, x_n)$ bomo, ko nas posamezne spremenljivke ne bodo zanimale, pisali $f(\vec{x})$, kjer je $\vec{x} = (x_1, x_2, \ldots, x_n)$. Podobno bo za lambda izraze $\lambda\vec{x}.M$ pomenilo $\lambda x_1\, x_2 \ldots x_n.M$.

Večina teorije in terminologije o rekurzivnih funkcijah je vzete iz knjige [Pri94].

3.1. **Primitivno rekurzivne funkcije.**

Definicija 3.1. *Začnimo z definicijo* primitivno rekurzivnih *funkcij.*

(1) *Enomestna funkcija $\mathcal{Z}$, za katero velja*

$$\forall x \in \mathbb{N}_0 : \mathcal{Z}(x) = 0,$$

je primitivno rekurzivna. Imenujemo jo ničelna *funkcija.*

(2) *Enomestna funkcija $\mathcal{N}$, za katero velja*

$$\forall x \in \mathbb{N}_0 : \mathcal{N}(x) = x + 1,$$

je primitivno rekurzivna. Imenujemo jo nasledstvena *funkcija ali funkcija* naslednik.

(3) *n-mestna funkcija $\mathcal{P}_i^n$, kjer je $n \in \mathbb{N}$ in $1 \leq i \leq n$, za katero velja*

$$\forall (x_1, x_2, \ldots, x_n) \in \mathbb{N}_0^n : \mathcal{P}_i^n(x_1, x_2, \ldots, x_n) = x_i,$$

je primitivno rekurzivna. Imenujemo jo projekcijska funkcija *ali* projekcija.

(4) *Naj bodo m-mestna funkcija $\boldsymbol{g}$ in $\boldsymbol{n}$-mestne funkcije $\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_m}$ primitivno rekurzivne. Potem je tudi n-mestna funkcija $\boldsymbol{f}$, definirana s predpisom*

$$\forall \vec{x} \in \mathbb{N}_0^n : \boldsymbol{f}(\vec{x}) = \boldsymbol{g}(\boldsymbol{h_1}(\vec{x}), \boldsymbol{h_2}(\vec{x}), \ldots, \boldsymbol{h_m}(\vec{x}))$$

primitivno rekurzivna. Za tako definirano funkcijo $\boldsymbol{f}$ pravimo, da smo jo dobili iz funkcij $\boldsymbol{g}, \boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_m}$ s substitucijo.

(5) *Naj bosta n-mestna funkcija $\boldsymbol{g}$ in $(n+2)$-mestna funkcija $\boldsymbol{h}$ primitivno rekurzivni. Potem je rekurzivna tudi $(n+1)$-mestna funkcija $\boldsymbol{f}$, definirana s predpisoma $\forall \vec{x} \in \mathbb{N}_0^n \forall y \in \mathbb{N}_0$:*

$$\boldsymbol{f}(0, \vec{x}) = \boldsymbol{g}(\vec{x})$$
$$\boldsymbol{f}(y+1, \vec{x}) = \boldsymbol{h}(y, \boldsymbol{f}(y, \vec{x}), \vec{x})$$

Za tako definirano funkcijo $\boldsymbol{f}$ pravimo, da smo jo dobili iz funkcij $\boldsymbol{g}$ in $\boldsymbol{h}$ z uporabo primitivne rekurzije. *Spremenljivke $x_1, x_2, \ldots, x_n$ imenujemo* parametri *rekurzije.*

V posebnem primeru, ko je $n = 0$, dobimo rekurzijo brez parametrov. Naj bo dvomestna funkcija $\boldsymbol{h}$ primitivno rekurzivna in k izbrano naravno število. Potem je primitivno rekurzivna tudi funkcija $\boldsymbol{f}$ definirana s predpisoma

$$\boldsymbol{f} = k$$
$$\boldsymbol{f}(y+1) = \boldsymbol{h}(y, f(y)).$$

Funkcijam iz točk (1), (2) *in* (3) *pravimo* začetne *funkcije.*

Da je funkcija primitivno rekurzivna lahko povemo tudi tako: funkcija je primitivno rekurzivna *natanko tedaj, ko jo lahko predstavimo kot zadnji člen končnega zaporedja primitivno rekurzivnih funkcij, pri katerem je vsak člen ali začetna funkcija ali pa je narejen iz členov zaporedja pred njim z uporabo substitucije ali primitivne rekurzije. Jasno je, da je prvi člen zaporedja začetna funkcija.*

Iz definicije je razvidno, da če hočemo pokazati, da imajo vse primitivno rekurzivne funkcije neko lastnost, je dovolj pokazati, da imajo to lastnost vse začetne funkcije in da se lastnost ohranja pri substituciji in primitivni rekurziji.

Trditev 3.2. *Vsaka n-mestna primitivna rekurzivna funkcija je totalna.*

Dokaz. Po definiciji trditvi ustrezajo začetne funkcije. Trditev je jasna tudi za substitucijo. Dokažimo še, da je funkcija definirana z uporabo primitivne rekurzije res totalna.

Naj bo $\boldsymbol{g}$ n-mestna totalna funkcija in $\boldsymbol{h}$ $(n+2)$-mestna totalna funkcija. Dokazujemo z indukcijo na y. Če je $y = 0$ je $\boldsymbol{f}(0, \vec{x}) = \boldsymbol{g}(\vec{x})$, torej definirana za vsako n-terico $\vec{x}$ in zato totalna.

Predpostavimo, da je $\boldsymbol{f}(y, \vec{x})$ definiran za vsako n-terico $\vec{x}$. Potem je, po definiciji, $\boldsymbol{f}(y+1, \vec{x}) = \boldsymbol{h}(y, \boldsymbol{f}(y, \vec{x}), \vec{x})$. Ker je $\boldsymbol{h}$ totalna, za $\boldsymbol{f}$ uporabimo indukcijsko predpostavko in smo končali. Funkcija $\boldsymbol{f}$ je definirana za vsako n-terico $\vec{x}$ in za vsak $y \in \mathbb{N}_0$, torej totalna. □

Kmalu so našli funkcije, ki so intuitivno izračunljive, torej lahko zapišemo recept za izračun, vendar pa niso primitivno rekurzivne. V resnici je, z varianto Cantorjevega diagonalnega argumenta, z malo teorije rekurzivnih funkcij, lahko pokazati da obstajajo intuitivno izračunljive funkcije, ki pa niso primitivno rekurzivne. Med prvimi znanimi, za katere znamo konkretno zapisati predpis, je bila Ackermannova funkcija, definirana s predpisi

$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0 \wedge n = 0 \\ A(m-1, A(m, n-1)) & m > 0 \wedge n > 0 \end{cases}.$$

Dokaz dejstva, da ta funkcija ni primitivno rekurzivna zopet ni bistven, je pa dolg in ni v okviru tega dela. Glavna težava je, da vrednosti $A(m,n)$ zelo hitro rastejo, hitreje kot katerakoli primitivno rekurzivna funkcija. $A(4,2)$ je na primer $2^{65536} - 3$. Po drugi strani pa je predpis za izračun dovolj lep, da bi vsak, z dovolj časa, znal izračunati vrednosti funkcije za poljubne začetne vrednosti samo z uporabo funkcij naslednik in predhodnik.

Primeri 3.3. *Oglejmo si nekaj primerov primitivno rekurzivnih funkcij:*

(1) *Enomestna funkcija* predhodnik *je primitivno rekurzivna. Res,*

$$\begin{aligned} \mathbf{predhodnik}(0) &= 0 \\ \mathbf{predhodnik}(y+1) &= \mathcal{P}_1^2(y, \mathbf{predhodnik}(y)). \end{aligned}$$

(2) *Dvomestna funkcija vsota je primitivno rekurzivna. Res, definiramo jo lahko z uporabo primitivne rekurzije:*

$$\begin{aligned} \mathbf{vsota}(0,x) &= \mathcal{P}_1^1(x) \\ \mathbf{vsota}(y+1,x) &= \mathcal{N}(\mathcal{P}_2^3(y, \mathbf{vsota}(y,x), x)). \end{aligned}$$

V tem primeru sta funkciji $\boldsymbol{g}$ in $\boldsymbol{h}$ iz definicije primitivne rekurzije definirani s predpisoma $\boldsymbol{h}(x,y,z) = \mathcal{N}(\mathcal{P}_2^3(x,y,z))$ in $\boldsymbol{g}(x) = \mathcal{P}_1^1(x)$. Funkcija g je primitivno rekurzivna ker je začetna, h pa ker je narejena z uporabo substitucije iz začetnih funkcij.

(3) *Dvomestna funkcija* produkt *je primitivno rekurzivna. Podobno kot vsoto jo lahko definiramo z uporabo primitivne rekurzije:*

$$\begin{aligned} \mathbf{produkt}(0,x) &= \mathcal{Z}(x) \\ \mathbf{produkt}(y+1,x) &= \mathbf{vsota}(\mathcal{P}_2^3(y, \mathbf{produkt}(y,x), x), \mathcal{P}_3^3(y, \mathbf{produkt}(y,x), x)). \end{aligned}$$

(4) *Enomestna funkcija fakulteta je primitivno rekurzivna.*

$$\mathbf{fakulteta}(0) = 1$$
$$\mathbf{fakulteta}(y+1) = \mathbf{produkt}(\mathcal{P}_2^2(y, \mathbf{fakulteta}(y)), \mathcal{N}(\mathcal{P}_1^2(y, \mathbf{fakulteta}(y)))).$$

3.2. **Primitivno rekurzivne funkcije so lambda izrazljive.** Takoj pokažimo, da lahko vse primitivno rekurzivne funkcije izrazimo v lambda računu. Pokazali bomo navidezno malo bolj strogo trditev, da vse primitivno rekurzivne funkcije lahko izrazimo z zaprtimi lambda izrazi.

Lema 3.4. *Začetne funkcije so lambda izrazljive z zaprtimi izrazi.*

Dokaz. Spomnimo se, da v definiciji lambda izrazljivih funkcij za predstavitev naravnih števil uporabljamo Scottove numerale.

Ničelna funkcija: Ustrezen izraz smo že definirali: $\mathbf{Z} \equiv \lambda x.\overline{0}$. Ker je $\overline{0}$ zaprt izraz po definiciji β-redukcije in lemi 2.10 velja $\mathbf{Z}M \rhd_\beta \overline{0}$ za vsak lambda izraz M, torej v posebnem tudi za vse Scottove numerale.

Nasledstvena funkcija: Ustrezen izraz smo že definirali: $\mathbf{N} \equiv \lambda n\, x\, y.y\, n$. Hkrati po lemi 2.28 sledi, da ta izraz lambda izraža funkcijo $\mathcal{N}$.

Projekcija: Definirajmo $\boldsymbol{P_i^n} \equiv \lambda x_1\, x_2\, \ldots\, x_n.x_i$. Preverimo, da za poljubne zaprte lambda izraze $M_1, M_2, \ldots, M_n$ velja

$$\boldsymbol{P_i^n}\, M_1\, M_2 \cdots M_n \rhd_\beta M_i.$$

Iz izraza $P_i^n M_1 M_2 \cdots M_n$ z n-kratno uporabo β-krčenj, z vmesno uporabo leme 2.13, dobimo

$$x_i\, [x_1 \to M_1]\, [x_2 \to M_2] \cdots [x_n \to M_n]\,.$$

Po posledici 2.12 je to enako M_i, saj so vsi M_i zaprti izrazi.

Po konstrukciji so tudi vsi lambda izrazi zaprti. □

Lema 3.5 (Substitucija je lambda izrazljiva)**.** *Naj bodo m-mestna funkcija $\boldsymbol{g}$ in n-mestne funkcije $\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_m}$ lambda izrazljive z izrazi $\overline{g}, \overline{h_1}, \overline{h_2}, \ldots, \overline{h_n}$. Potem je lambda izrazljiva tudi funkcija $\boldsymbol{f}(\vec{x}) = \boldsymbol{g}(\boldsymbol{h_1}(\vec{x}), \boldsymbol{h_2}(\vec{x}), \ldots, \boldsymbol{h_m}(\vec{x}))$ in to z izrazom*

$$\overline{f} \equiv \lambda\vec{x}.\overline{g}(\overline{h_1}\vec{x})(\overline{h_2}\vec{x}) \cdots (\overline{h_m}\vec{x}).$$

Če so izrazi $\overline{g}, \overline{h_1}, \overline{h_2}, \ldots, \overline{h_n}$ zaprti, je zaprt tudi izraz $\overline{f}$.

Dokaz. Dokazati moramo, da velja $\overline{f}\,\overline{n_1}\,\overline{n_2} \cdots \overline{n_n} =_\beta \overline{\boldsymbol{f}(n_1, n_2, \ldots, n_n)}$.

Vemo, da velja $\overline{g}\,\overline{n_1}\,\overline{n_2}\, \ldots\, \overline{n_m} =_\beta \overline{\boldsymbol{g}(n_1, n_2, \ldots, n_m)}$ za vsak nabor $n_1, n_2, \ldots, n_m$ in $\overline{h_i}\,\overline{n_1}\,\overline{n_2}\, \ldots\, \overline{n_n} =_\beta \overline{\boldsymbol{h_i}(n_1, n_2, \ldots, n_n)}$ za vsak $i \in \{1, 2, \ldots, n\}$ in vsak nabor $n_1, \ldots, n_n$. Iskana enakost sedaj sledi iz definicije substitucije na aplikaciji ob uporabi lem 2.10 in 2.13.

Izraz $\overline{f}$ je tudi očitno zaprt, če so le vsi izrazi $\overline{g}, \overline{h_1}, \overline{h_2}, \ldots, \overline{h_m}$ zaprti. □

Za izražavo primitivne rekurzije potrebujemo še nekaj pomožne teorije. Težava je, da v predpisih za funkcijo $\boldsymbol{f}$ le ta nastopa na levi in desni strani enakosti. Težavo bomo rešili tako, da bomo funkcije definirane s primitivno rekurzijo našli kot negibne točke nekaterih lambda izrazov.

Lema 3.6. *Obstaja lambda izraz $\mathbf{Y}$, da za vsak lambda izraz M velja*

$$\mathbf{Y}M \rhd_\beta M(\mathbf{Y}M).$$

Dokaz. Možnosti za izbiro izraza $\mathbf{Y}$ je več. Izberimo

$$\mathbf{Y} \equiv UU,$$

kjer je $U \equiv \lambda ux.x(uux)$. Preverimo, da je ustrezen. Naj bo M poljuben lambda izraz:

$$\mathbf{Y}M \equiv (\lambda ux.x(uux))UM$$

in če dvakrat uporabimo β-krčenje in upoštevamo, da je izraz U zaprt, dobimo

$$\rhd_\beta M(UUM),$$

kar je po definiciji izraza $\mathbf{Y}$ enako

$$\equiv M(\mathbf{Y}M). \qquad \square$$

Iz leme sledi, da ima vsak lambda izraz negibno točko, t.j. za vsak izraz M obstaja izraz N, da je $MN =_\beta N$. Res, za N lahko vzamemo kar $\mathbf{Y}M$.

Posledica 3.7. *Za vsak lambda izraz M, nabor spremenljivk $x, y_1, y_2, \ldots, y_n$, $n \in \mathbb{N}_0$ in lambda izraze $N_1, N_2, \ldots, N_n$, v katerih $y_1, y_2, \ldots, y_n$ ne nastopajo prosto, obstaja izraz N, v katerem spremenljivke $y_1, y_2, \ldots, y_n$ ne nastopajo prosto, da je*

$$NN_1N_2 \ldots N_n \rhd_\beta M\,[x \to N]\,[y_1 \to N_1]\,[y_2 \to N_2] \cdots [y_n \to N_n]\,.$$

Dokaz. Vzamemo $N \equiv \mathbf{Y}(\lambda xy_1y_2 \ldots y_n.M)$.

$$NN_1N_2 \ldots N_n \rhd_\beta (\lambda xy_1y_2 \ldots y_n.M)NN_1N_2 \ldots N_n$$

po lastnosti izraza $\mathbf{Y}$. Če sedaj uporabimo β-krčenje in upoštevamo, da noben od $y_1, y_2, \ldots y_n$ ni prost v N, torej lahko uporabimo lemo 2.13, dobimo

$$\rhd_\beta (\lambda y_1y_2 \ldots y_n.M\,[x \to N])N_1N_2 \ldots N_n.$$

Trditev sledi po indukciji na n ob uporabi leme 2.13. $\square$

Posledica je bolj uporabna od leme, saj bomo z njeno uporabo lahko enostavno definirali izraze, ki bodo predstavljali funkcije, definirane z uporabo primitivne rekurzije. Res, izraz N bo postal funkcija $\boldsymbol{f}$ iz definicije primitivne rekurzije, M bo izraz, v katerem bodo nastopali lambda izrazi, ki predstavljajo funkciji $\boldsymbol{g}$ in $\boldsymbol{h}$, $N_1, N_2, \ldots, N_n$ pa bodo Scottovi numerali.

Definicija 3.8. *Definirajmo lambda izraze*

$$\mathbf{T} \equiv \overline{0} \equiv \lambda xy.x,$$
$$\mathbf{F} \equiv \lambda xy.y$$

in

$$\mathbf{D} \equiv \lambda x.x\mathbf{T}(\lambda y.\mathbf{F}).$$

Lema 3.9. *Velja:*

(1) $\forall n \in \mathbb{N} : \mathbf{D}\,\overline{n} \rhd_\beta \mathbf{F}$ *in* $\mathbf{D}\,\overline{0} \rhd_\beta \mathbf{T}$.

(2) *Naj bosta M in N lambda izraza, v katerih spremenljivki x in y ne nastopata prosto. Potem velja* $\mathbf{T}\,M\,N \rhd_\beta M$ *in* $\mathbf{F}\,M\,N \rhd_\beta N$.

Dokaz. Naj bo najprej $n \in \mathbb{N}$.

$$\begin{aligned}\mathbf{D}\,\overline{n} &\equiv (\lambda x.x\,\mathbf{T}\,(\lambda y.\mathbf{F}))\,(\lambda x\,y.y\,\overline{n-1}) \\ &\rhd_\beta (\lambda x\,y.y\,\overline{n-1})\,\mathbf{T}\,(\lambda y.\mathbf{F}) \\ &\rhd_\beta (\lambda y.\mathbf{F})\,\overline{n-1} \\ &\rhd_\beta \mathbf{F}.\end{aligned}$$

Preverimo še za 0.

$$\begin{aligned}\mathbf{D}\,\overline{0} &\equiv (\lambda x.x\,\mathbf{T}\,(\lambda y.\mathbf{F}))\,(\lambda x\,y.x) \\ &\rhd_\beta (\lambda x\,y.x)\,\mathbf{T}\,(\lambda y.\mathbf{F}) \\ &\rhd_\beta \mathbf{T}.\end{aligned}$$

Podobno preverimo drugo točko leme. □

Lema 3.10 (Primitivna rekurzija je lambda izrazljiva)**.** *Naj bosta n-mestna funkcija $\boldsymbol{g}$ in $(n+2)$-mestna funkcija $\boldsymbol{h}$ lambda izrazljivi primitivno rekurzivni funkciji. Potem je lambda izrazljiva tudi $(n+1)$-mestna funkcija $\boldsymbol{f}$, definirana s predpisoma*

$$\begin{aligned}\boldsymbol{f}(0,\vec{x}) &= \boldsymbol{g}(\vec{x}) \\ \boldsymbol{f}(y+1,\vec{x}) &= \boldsymbol{h}(y,\boldsymbol{f}(y,\vec{x}),\vec{x}).\end{aligned}$$

Če sta izraza $\overline{g}$ in $\overline{h}$ zaprta, je zaprt tudi izraz $\overline{f}$.

Dokaz. Definirajmo

$$\overline{f} \equiv \lambda y\,\vec{x}.(\mathbf{R}\,y(\overline{g}\,\vec{x})(\lambda uv.\overline{h}\,uv\vec{x})),$$

kjer je $\mathbf{R}$ zaprt izraz, za katerega za vse zaprte lambda izraze M in N ter numerale $\overline{k}$ velja:

$$\begin{aligned}\mathbf{R}\,\overline{0}MN &\rhd_\beta M, \\ \mathbf{R}\,\overline{(k+1)}MN &\rhd_\beta N\overline{k}(\mathbf{R}\,\overline{k}MN).\end{aligned}$$

Če tak $\mathbf{R}$ obstaja smo našli ustrezen lambda izraz, s katerim lahko izrazimo primitivno rekurzijo. Res, velja

$$\overline{f}\,\overline{0}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n} \rhd_\beta \mathbf{R}\,\overline{0}\,(\overline{g}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n})(\lambda uv.\overline{h}\,u\,v\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n})$$

po definiciji substitucije na aplikaciji, β redukcije in lemah 2.10 in 2.13, od tod pa sledi

$$\rhd_\beta \overline{g}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n}$$

po lastnostih izraza $\mathbf{R}$.

Preverimo še za $k+1$.

$$\overline{f}\,\overline{(k+1)}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n} \rhd_\beta \mathbf{R}\,\overline{(k+1)}(\overline{g}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n})(\lambda uv.\overline{h}\,u\,v\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n})$$

zopet po definiciji substitucije na aplikaciji, β redukcije in lemah 2.10 in 2.13. Po zahtevani lastnosti izraza $\mathbf{R}$ sledi

$$\rhd_\beta (\lambda uv.\overline{h}\,u\,v\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n})\,\overline{k}\,(\mathbf{R}\,\overline{k}\,(\overline{g}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n})(\lambda uv.\overline{h}\,u\,v\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n}))$$

in če uporabimo definicijo izraza $\overline{f}$, definicijo substitucije, β-redukcije in lemi 2.10 ter 2.13, dobimo

$$\rhd_\beta (\lambda uv.\overline{h}\, u\, v\, \overline{n_1}\, \overline{n_2}\, \ldots\, \overline{n_n})\, \overline{k}\, (\overline{f}\, \overline{k}\, \overline{n_1}\, \overline{n_2}\, \ldots\, \overline{n_n})$$

in če sedaj dvakrat uporabimo β-krčenje in vmes lemo 2.13, dobimo

$$\rhd_\beta \overline{h}\, \overline{k}\, (\overline{f}\, \overline{k}\, \overline{n_1}\, \overline{n_2}\, \ldots\, \overline{n_n})\, \overline{n_1}\, \overline{n_2}\, \ldots\, \overline{n_n},$$

kar je želeni rezultat.

Skonstruirajmo sedaj še $\mathbf{R}$. Zadoščati mora zvezi

$$\mathbf{R}\, \overline{k}\, M\, N \rhd_\beta \mathbf{D}\, \overline{k}\, M\, (N\, (\mathbf{P}\, \overline{k})(\mathbf{R}\, (\mathbf{P}\, \overline{k})\, M\, N)). \tag{1}$$

Po posledici 3.7 za izraz $\mathbf{D}\, y_1\, y_2\, (y_3\, (\mathbf{P}\, y_1)(x\, (\mathbf{P}\, y_1)\, y_2\, y_3))$ obstaja lambda izraz, poimenujmo ga $\mathbf{R}$, ki zadošča zvezi 1. Znamo ga tudi skonstruirati:

$$\mathbf{R} \equiv \mathbf{Y}(\lambda x\, y_1\, y_2\, y_3.(\mathbf{D}\, y_1\, y_2\, (y_3\, (\mathbf{P}\, y_1)(x\, (\mathbf{P}\, y_1)\, y_2\, y_3)))).$$

Enostavno je preveriti, da definirani izraz res zadošča zahtevanima lastnostima. Očitno je $\mathbf{R}$ zaprt, saj so izrazi $\mathbf{D}$, $\mathbf{P}$ in $\mathbf{Y}$ zaprti, spremenljivke x, y_1, y_2 in y_3 pa so tudi vezane v izrazu $\mathbf{R}$. Od tod sledi, da je tudi izraz $\overline{f}$ zaprt, če sta le izraza $\overline{g}$ in $\overline{h}$ zaprta. □

Za trenutek se ustavimo in si oglejmo konkretni primer pretvorbe primitivno rekurzivne funkcije v ustrezen lambda izraz. Funkciji naslednik in predhodnik smo že predstavili v lambda računu. Poglejmo si kaj dobimo, če naredimo konstrukcijo funkcije predhodnik s pravkar pokazanimi metodami. Označimo jo s $\boldsymbol{P'}$.

Funkcijo predhodnik smo v 3.3 definirali s primitivno rekurzijo, zato $\boldsymbol{P'}$ definiramo kot v dokazu leme 3.10 z izrazom

$$\boldsymbol{P'} \equiv \lambda y.\mathbf{R}\, y\, \overline{0}\, (\lambda u\, v.(\lambda x\, y.x)\, u\, v).$$

Upoštevajmo definicijo $\mathbf{R}$ in poenostavimo dobljeni izraz. Najprej poenostavimo $(\lambda u\, v.(\lambda x\, y.x)\, u\, v)$ in dobimo

$$\rhd_\beta \lambda y.\mathbf{R}\, y\, \overline{0}\, (\lambda u\, v.u).$$

Sedaj razpišemo $\mathbf{R}$

$$\rhd_\beta \lambda y.\mathbf{Y}(\lambda x\, y_1\, y_2\, y_3.(\mathbf{D}\, y_1\, y_2\, (y_3\, (\mathbf{P}\, y_1)(x\, (\mathbf{P}\, y_1)\, y_2\, y_3))))\, y\, \overline{0}\, (\lambda u\, v.u)$$

in upoštevamo lastnost izraza $\mathbf{Y}$, da dobimo

$$\rhd_\beta \lambda y.(\lambda x\, y_1\, y_2\, y_3.(\mathbf{D}\, y_1\, y_2\, (y_3\, (\mathbf{P}\, y_1)(x\, (\mathbf{P}\, y_1)\, y_2\, y_3))))\, (\mathbf{Y}\, M)\, y\, \overline{0}\, (\lambda u\, v.u),$$

kjer smo z M označili $\lambda x\, y_1\, y_2\, y_3.(\mathbf{D}\, y_1\, y_2\, (y_3\, (\mathbf{P}\, y_1)(x\, (\mathbf{P}\, y_1)\, y_2\, y_3)))$. Sedaj uporabimo β-redukcijo in dobimo

$$\rhd_\beta \lambda y.(\mathbf{D}\, y\, \overline{0}\, ((\lambda u\, v.u)\, (\mathbf{P}\, y)((\mathbf{Y}\, M)\, (\mathbf{P}\, y)\, \overline{0}\, (\lambda u\, v.u))))$$
$$\rhd_\beta \lambda y.(\mathbf{D}\, y\, \overline{0}\, (\mathbf{P}\, y)).$$

Dobljeni izraz ni povsem enak izrazu $\mathbf{P}$, vendar pa iz lastnosti izraza $\mathbf{D}$ sledi, da res izraža funkcijo predhodnik. Podobno lahko definiramo vsoto, produkt in ostale

primitivno rekurzivne funkcije. Kot smo videli na primeru funkcije predhodnik, izrazi hitro postajajo zapleteni.

3.3. **Minimizacija.** Omenili smo že, da obstajajo enostavne funkcije, za katere znamo napisati algoritem, vendar niso primitivno rekurzivne, zato potrebujemo še eno operacijo na funkcijah, da bomo iz primitivno rekurzivnih funkcij dobili večji razred.

Definicija 3.11. *Če je* $\boldsymbol{g} : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ *funkcija z lastnostjo, da za vsak* $\vec{x} \in \mathbb{N}_0^n$ *obstaja* $y \in \mathbb{N}_0$*, da je* $\boldsymbol{g}(\vec{x}, y) = 0$*, lahko definiramo novo funkcijo* $\boldsymbol{f} : \mathbb{N}_0^n \to \mathbb{N}_0$ *s predpisom*

$$\boldsymbol{f}(\vec{x}) = \mu y(\boldsymbol{g}(\vec{x}, y) = 0) = \min_y \{y \mid \boldsymbol{g}(\vec{x}, y) = 0\}.$$

Operator μ *imenujemo* operator minimizacije. *Ker so naravna števila za standardno relacijo urejenosti dobro urejena, je definicija dobra in dobljena funkcija totalna.*

Definicija 3.12. *Funkcija je totalna rekurzivna, če jo lahko dobimo kot končni člen zaporedja funkcij, tako da je vsak člen zaporedja ali začetna funkcija ali pa je dobljen iz prejšnjih členov z uporabo substitucije, primitivne rekurzije ali minimizacije.*

Drugače povedano so totalne rekurzivne funkcije najmanjši razred numeričnih funkcij, ki vsebujejo začetne funkcije in so zaprte za primitivno rekurzijo, substitucijo in minimizacijo.

Lema 3.13 (Minimizacija je lambda izrazljiva)**.** *Naj bo* $\boldsymbol{g} : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ *funkcija z lastnostjo, da za vsak* $\vec{x} \in \mathbb{N}_0^n$ *obstaja* $y \in \mathbb{N}_0$*, da je* $\boldsymbol{g}(\vec{x}, y) = 0$*, lambda izrazljiva z lambda izrazom* $\overline{g}$*. Potem je lambda izrazljiva tudi funkcija* $\boldsymbol{f}(\vec{x}) = \mu y(\boldsymbol{g}(\vec{x}, y) = 0)$*. Če je izraz* $\overline{g}$ *zaprt, je zaprt tudi izraz* $\overline{f}$*.*

Dokaz. En način, kako izračunati $\mu y(g(\vec{x}, y) = 0)$ je, da najdemo izraz $\mathbf{H}$, ki "vrne" $\overline{y}$, če je $g(\vec{x}, y) = 0$, sicer pa nadaljuje z računanjem na $\overline{y+1}$. Lambda izraz $\mathbf{H}$ mora torej ustrezati zvezi

$$\mathbf{H}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n}\,\overline{y} \rhd_\beta \mathbf{D}\,(\overline{g}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n}\,\overline{y})\,\overline{y}(\mathbf{H}\,\overline{n_1}\,\overline{n_2}\,\ldots\,\overline{n_n}\,(\mathbf{N}\,\overline{y})).$$

Tak $\mathbf{H}$ obstaja po posledici 3.7 in je enak

$$\mathbf{H} \equiv \mathbf{Y}(\lambda u\,\vec{x}\,y.\mathbf{D}\,(\overline{g}\,\vec{x}\,y)\,y\,(u\,\vec{x}\,(\mathbf{N}\,y))).$$

Ko enkrat tak $\mathbf{H}$ imamo, je enostavno definirati $\overline{f}$:

$$f \equiv \lambda\vec{x}.\mathbf{H}\,\vec{x}\,\overline{0}.$$

Sedaj je enostavno pokazati, da definirani $\overline{f}$ res zadošča zahtevanim lastnostnim. □

S tem smo dokazali naslednji izrek.

Izrek 3.14. *Naj bo* $\boldsymbol{f}$ *totalna rekurzivna funkcija. Potem je* $\boldsymbol{f}$ *lambda izrazljiva.*

Velja tudi obratno, vse lambda izrazljive funkcije so totalne rekurzivne, vendar tega ne bomo dokazovali. Kratek dokaz, ki uporabi nekaj močnih izrekov iz teorije rekurzivnih funkcij, lahko najdemo v [Bar84].

3.4. **Delne rekurzivne funkcije.** Oglejmo si za trenutek še delne rekurzivne funkcije.

Definicija 3.15. *Delni numerični funkciji f in g sta enaki, pišemo $f \simeq g$, natanko tedaj, ko sta definirani na isti množici in tam sovpadata.*

Definicija 3.16. *Naj bo $\mathcal{R}$ razred delnih numeričnih funkcij.*

(1) *$\mathcal{R}$ je zaprt za substitucijo, če so vsi $\boldsymbol{f}$ definirani s predpisom*

$$\boldsymbol{f}(\vec{x}) \simeq \boldsymbol{g}(\boldsymbol{h_1}(\vec{x}), \boldsymbol{h_2}(\vec{x}), \ldots, \boldsymbol{h_m}(\vec{x})),$$

za $\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_m}, \boldsymbol{g} \in \mathcal{R}$, tudi v $\mathcal{R}$.

(2) *$\mathcal{R}$ je zaprt za minimizacijo, če so vsi $\boldsymbol{f}$ definirani s predpisom*

$$\boldsymbol{f}(\vec{x}) \simeq \mu y(\boldsymbol{g}(\vec{x}, y) = 0),$$

kjer je $\boldsymbol{g}$ totalna funkcija, tudi v $\mathcal{R}$.

V drugi točki ne zahtevamo, da tak y sploh obstaja in s tem lahko iz totalnih funkcij dobimo delne.

Definicija 3.17. *Razred delnih rekurzivnih funkcij je najmanjši razred funkcij, ki vsebuje totalne rekurzivne funkcije in je zaprt za prej definirani operaciji substitucije in minimizacije.*

Če hočemo v lambda računu predstaviti tudi delne rekurzivne funkcije moramo izbrati način, kako predstaviti rezultat funkcije, če funkcijo uporabimo na argumentih, ki niso v domeni funkcije. Možnih načinov je več, osnovni pa je, da tedaj izberemo izraz brez normalne oblike.

Definicija 3.18. *Delna numerična funkcija $\boldsymbol{f} : D \subseteq \mathbb{N}_0^n \to \mathbb{N}_0$ je lambda izrazljiva, če obstaja lambda izraz F, da je*

$$\forall (n_1, n_2, \ldots, n_n) \in D : F\,\overline{n_1}\,\overline{n_2} \cdots \overline{n_n} \rhd_\beta \overline{\boldsymbol{f}(n_1, n_2, \ldots, n_n)}$$
$$\forall (n_1, n_2, \ldots, n_n) \notin D : F\,\overline{n_1}\,\overline{n_2} \cdots \overline{n_n} \text{ nima normalne oblike}$$

Opazimo da se ta definicija lambda izrazljivih funkcij na totalnih funkcijah ujema s prejšnjo.

Dokaz, da so delne rekurzivne funkcije lambda izrazljive, je tehnično dosti bolj zahteven kot za totalne. Glavna težava je, da moramo za izraze pokazati da nimajo normalne oblike in to ni lahko brez Church-Rosserjevih izrekov. Dokaz je npr. v [Hin08] na straneh 58-60, vendar z uporabo še enega močnega izreka iz teorije rekurzivnih funkcij, Kleenejevega izreka o normalni obliki

Bolj koristno bo, če si pogledamo, kako lahko v lambda računu programiramo.

4. Lambda račun kot minimalističen programski jezik

Ta razdelek bo manj formalen od prejšnjih. Večinoma ne bomo rigorozno dokazovali, da definirani izrazi res zadoščajo zahtevam, vendar se bo to ponavadi videlo po konstrukciji. Definirali bomo dovolj izrazov, da bomo lahko na koncu zapisali algoritem hitrega urejanja v lambda računu.

4.1. **Osnovni podatkovni tipi.** Definirali smo že izraze, ki predstavljajo naravna števila in logične vrednosti, čeprav jih nismo eksplicitno tako poimenovali. **T** definiramo kot **True**, **F** pa kot **False**.

Definirajmo še izraze, ki nam bodo predstavljali sezname. Ideja je podobna kot pri definiciji naravnih števil. Seznam je prazen, ali pa je sestavljen iz glave in repa, kjer je glava neka vrednost, rep pa zopet seznam.

Prazen seznam definirajmo kot $\mathbf{nil} \equiv \mathbf{F}$, funkcijo, ki iz podane vrednosti in seznama zgradi nov seznam pa $\mathbf{cons} \equiv \lambda x\, y.\lambda z.z\, x\, y$. Podobno lahko definiramo pare $\mathbf{pair} \equiv \lambda x\, y.\lambda z.z\, x\, y$.

V obeh definicijah (in tudi v definiciji naravnih števil) se pred dejanskimi vrednostmi pojavlja spremenljivka, npr. z v definiciji **pair**. To ni naključje. Spremenljivka preprečuje, da bi nastali novi β-redeksi, in s tem ne bi mogli iz para ali seznama nazaj dobiti vrednosti.

4.2. **Operacije na osnovnih tipih.** Do sedaj nismo še nikjer pokazali, kako definiramo seštevanje na Scottovih numeralih.

Če uporabimo definicijo seštevanja s primitivno rekurzivnimi funkcijami in jo neposredno prevedemo v lambda račun, dobimo

$$\mathbf{plus} \equiv \lambda y\, x.\mathbf{R}\, y\, x\, (\lambda u\, v.\mathbf{N}\, v).$$

Podobno dobimo za množenje izraz

$$\mathbf{prod} \equiv \lambda y\, x.\mathbf{R}\, y\, \overline{0}\, (\lambda u\, v.\mathbf{plus}\, x\, v).$$

Nadaljujemo lahko z omejenim odštevanjem. Ker $n - m$ v naravnih številih ni definirano za $m > n$, v tem primeru definiramo $n - m = 0$. Definiramo ga skoraj enako kot seštevanje, le da uporabimo funkcijo predhodnik namesto funkcije naslednik. Dobimo

$$\mathbf{sub} \equiv \lambda y\, x.\mathbf{R}\, y\, x\, (\lambda u\, v.\mathbf{P}\, v).$$

Z metodami iz prejšnjih dokazov je lahko preveriti, da ima izraz res zahtevane lastnosti.

Ko imamo odštevanje, lahko definiramo relaciji večje ali enako in manjše ali enako:

$$\mathbf{geq} \equiv \lambda x\, y.\mathbf{D}\, (\mathbf{sub}\, y\, x) \qquad\qquad \mathbf{leq} \equiv \lambda x\, y.\mathbf{D}\, (\mathbf{sub}\, x\, y).$$

Preverimo za **geq**:

$m \geq n$**:**

$$\mathbf{geq}\, \overline{m}\, \overline{n} =_\beta \mathbf{D}\, (\mathbf{sub}\, \overline{n}\, \overline{m})$$

kar je po predpostavki na n in m ter naši definiciji odštevanja enako

$$=_\beta \mathbf{D}\, \overline{0} =_\beta \mathbf{T}.$$

$m < n$**:** Naj bo $m - n = k$. Po predpostavki je $k \geq 1$.

$$\mathbf{geq}\, \overline{m}\, \overline{n} =_\beta \mathbf{D}\, (\mathbf{sub}\, \overline{n}\, \overline{m})$$

kar je po predpostavki enako

$$=_\beta \mathbf{D}\, \overline{k}$$

kar pa je po lastnosti izraza **D** enako

$$=_\beta \mathbf{F}.$$

Definiramo lahko tudi stavek **if**, to je lambda izraz, za katerega je $\mathbf{if}\, B\, M\, N =_\beta M$, če je $B =_\beta \mathbf{T}$ in $\mathbf{if}\, B\, M\, N =_\beta N$, če je $B =_\beta \mathbf{F}$, za vse zaprte lambda izraze M, N. Definicija je zelo enostavna,

$$\mathbf{if} \equiv \lambda x\, y\, z.x\, y\, z.$$

Preverimo:

$B =_\beta \mathbf{T}$:

$$\begin{aligned}\mathbf{if}\, B\, M\, N &=_\beta B\, M\, N\\ &=_\beta \mathbf{T}\, M\, N\end{aligned}$$

kar je po lastnosti izraza **T** enako

$$=_\beta M.$$

$B =_\beta \mathbf{F}$:

$$\begin{aligned}\mathbf{if}\, B\, M\, N &=_\beta B\, M\, N\\ &=_\beta \mathbf{F}\, M\, N\end{aligned}$$

kar je po lastnosti izraza **F** enako

$$=_\beta N.$$

V dokazu lahko opazimo, da bi lahko **if** definirali s še enostavnejšim izrazom;

$$\mathbf{if} \equiv \lambda x.x.$$

Logične veznike $\wedge$, $\vee$ in negacijo lahko sedaj enostavno definiramo z izrazi

$$\mathbf{and} \equiv \lambda x\, y.\mathbf{if}\, x\, y\, x \qquad \mathbf{or} \equiv \lambda x\, y.\mathbf{if}\, x\, x\, y \qquad \mathbf{not} \equiv \lambda x.\mathbf{if}\, x\, \mathbf{F}\, \mathbf{T}.$$

Z logičnimi vezniki lahko združimo funkcije **geq** in **leq**, da dobimo izraz za relacijo enakosti, relacijo strogo manjši in relacijo strogo večji. Ustrezni izrazi so

$$\begin{aligned}\mathbf{lt} &\equiv \lambda n\, m.\mathbf{not}\, (\mathbf{geq}\, n\, m),\\ \mathbf{gt} &\equiv \lambda n\, m.\mathbf{not}\, (\mathbf{leq}\, n\, m)\end{aligned}$$

in

$$\mathbf{eq} \equiv \lambda n\, m.\mathbf{and}\, (\mathbf{leq}\, n\, m)(\mathbf{geq}\, n\, m).$$

Oglejmo si še pare. Osnovni operaciji na parih sta **fst**, ki iz podanega para izlušči prvo komponento in **snd**, ki naredi enako za drugo komponento. Izraza lahko definiramo s

$$\mathbf{fst} \equiv \lambda x.x\, \mathbf{T}$$

in

$$\mathbf{snd} \equiv \lambda x.x\, \mathbf{F}.$$

Preverimo da **fst** deluje pravilno:

$$\begin{aligned}\mathbf{fst}\, (\mathbf{pair}\, M\, N) &=_\beta (\mathbf{pair}\, M\, N)\, \mathbf{T}\\ &\equiv (\lambda a\, b.\lambda z.z\, a\, b)\, M\, N\, \mathbf{T}\\ &=_\beta \mathbf{T}\, M\, N =_\beta M.\end{aligned}$$

Podobno imamo na seznamih definirani funkciji **head** in **tail**, ki iz danega seznama izluščita njegovo glavo in rep. Ustrezna izraza sta

$$\mathbf{head} \equiv \lambda x.\mathbf{fst}\, x \qquad\qquad \mathbf{tail} \equiv \lambda x.\mathbf{snd}\, x.$$

Od osnovnih funkcij za delo s seznami nam manjka še funkcija, ki ugotovi ali je seznam prazen ali ne, to je funkcija **null**, za katero velja $\mathbf{null}\ \mathbf{nil} =_\beta \mathbf{T}$ in $\mathbf{null}\, S =_\beta \mathbf{F}$, kjer je S poljuben neprazen seznam. Ustrezen lambda izraz je

$$\mathbf{null} \equiv \lambda x.x(\lambda a\, b.\mathbf{not})\mathbf{T}.$$

Definicija seznamov je rekurzivna, zato bo večina funkcij za delo s seznami rekurzivnih. Poglejmo si, kako lahko sistematično predstavimo rekurzivne funkcije v lambda računu. Vzemimo kar konkreten primer funkcije fakulteta, ki pa vsebuje osnovno idejo.

Izraz **fact** mora zadoščati zvezi

$$\mathbf{fact} =_\beta \lambda n.\mathbf{D}\, n\, \overline{1}\, (\mathbf{fact}\, (\mathbf{P}\, n))$$

in če sedaj abstrahiramo **fact** dobimo

$$= (\lambda f.\lambda n.\mathbf{D}\, n\, \overline{1}\, (f\, (\mathbf{P}\, n)))\, \mathbf{fact}.$$

Funkcija **fact** je torej negibna točka izraza $F \equiv (\lambda f.\lambda n.\mathbf{D}\, n\, \overline{1}\, (f\, (\mathbf{P}\, n)))$. Negibno točko izraza F pa znamo po posledici leme 3.6 zapisati kot $\mathbf{fact} \equiv \mathbf{Y}\, F$. Tako smo dobili lambda izraz, ki izraža funkcijo fakulteta. Potrebno je preveriti, da definirani izraz res zadošča zahtevam, saj ne vemo, ali je negibna točka $\mathbf{Y}\, F$ res želena. Na podoben način lahko definiramo tudi druge rekurzivne funkcije z lambda izrazi.

Sedaj si poglejmo še operacije na seznamih. Najprej definirajmo dve zelo uporabni funkciji na seznamih, **foldl** in **foldr**, s pomočjo katerih bomo lahko definirali skoraj vse potrebne funkcije. Naj bo $S = a_1, a_2, a_3, \ldots, a_n$ seznam. Potem želimo, da bi za **foldl** in **foldr** veljalo

$$\mathbf{foldl}\, f\, z\, S = f(\cdots(f(f(z, a_1), a_2), \cdots), a_n)$$

in

$$\mathbf{foldr}\, f\, z\, S = f(a_1, f(a_2, f(a_3, f(\cdots, f(a_n, z)\cdots)))).$$

Očitno morata funkciji zadoščati zvezi

$$\mathbf{foldl}\, f\, z\, s = \mathbf{if}\, (\mathbf{null}\, s)\, z\, (\mathbf{foldl}\, f\, (f\, z\, (\mathbf{head}\, s))\, (\mathbf{tail}\, s))$$

in

$$\mathbf{foldr}\, f\, z\, s = \mathbf{if}\, (\mathbf{null}\, s)\, z\, (f\, (\mathbf{head}\, s)\, (\mathbf{foldr}\, f\, z\, (\mathbf{tail}\, s))).$$

Izraze, ki takim zvezam zadoščajo pa že znamo poiskati. Definirajmo

$$\mathbf{foldl} \equiv \mathbf{Y}\, (\lambda x\, f\, z\, s.\mathbf{if}\, (\mathbf{null}\, s)\, z\, (x\, f\, (f\, z\, (\mathbf{head}\, s))\, (\mathbf{tail}\, s)))$$

in

$$\mathbf{foldr} \equiv \mathbf{Y}\, (\lambda x\, f\, z\, s.\mathbf{if}\, (\mathbf{null}\, s)\, z\, (f\, (\mathbf{head}\, s)\, (x\, f\, z\, (\mathbf{tail}\, s)))).$$

Da izrazi res ustrezajo zahtevanim zvezam sledi iz posledice 3.7. Poglejmo si, zakaj sta ti dve funkciji dobri. Vsoto seznama S lahko dobimo enostavno z izrazom $\mathbf{foldl}\ \mathbf{plus}\, \overline{0}\, S$, produkt s $\mathbf{foldl}\ \mathbf{prod}\, \overline{1}\, S$ in tako naprej. Enako bi lahko v teh dveh

primerih uporabili funkcijo **foldr**. Da funkciji res definirata vsoto in produkt elementov seznama se je najlažje prepričati s primerom. Dokaz ne bo ničesar razsvetlil, zato ga izpustimo. Na poti k definiciji hitrega urejanja potrebujemo še dve funkciji. Stik seznamov, **append**, in funkcijo **filter**, ki iz danega seznama izlušči samo elemente, ki zadoščajo danemu predikatu. Stik lahko definiramo z izrazom

$$\mathbf{append} \equiv \lambda x\, y.\mathbf{foldr}\ \mathbf{cons}\, y\, x,$$

filter pa z

$$\mathbf{filter} \equiv \lambda p\, s.\mathbf{foldr}\, (\lambda x\, y.\mathbf{if}\, (p\, x)\, (\mathbf{cons}\, x\, y)\, y)\, \mathbf{nil}\, s.$$

Za razumevanje teh dveh definicij si je smiselno predstavljati funkcijo f, ki jo podamo kot prvi argument funkciji **foldr** kot funkcijo, katere prvi argument je vedno trenutni člen seznama, drugi argument pa je rezultat funkcije, ki jo uporabimo na preostalih elementih seznama. Argument z funkcije **foldr** si lahko predstavljamo, kot da ga dodamo na konec seznama na nek način, odvisno od funkcije.

Oglejmo si sedaj hitro urejanje. Hitro urejanje je zelo preprost algoritem za urejanje seznama. Vzamemo nek element seznama, ki ga imenujemo pivot, in preostali del seznama razdelimo na dva dela. V prvem so elementi manjši od pivota, v drugem pa večji. Z hitrim urejanjem uredimo oba podseznama, potem pa oba seznama in pivot združimo nazaj v nov, urejen seznam. Ustrezen lambda izraz je

$$\begin{aligned}\mathbf{sort} \equiv \mathbf{Y}(\lambda f\, s.\mathbf{if}\, (\mathbf{null}\, s)\, \mathbf{nil}\, (\mathbf{append}\, (f\, (\mathbf{filter}\, (\mathbf{geq}\, (\mathbf{head}\, s))\, (\mathbf{tail}\, s)))\\ (\mathbf{cons}\, (\mathbf{head}\, s)\, (f\, (\mathbf{filter}\, (\mathbf{lt}\, (\mathbf{head}\, s))\, (\mathbf{tail}\, s)))))).\end{aligned}$$

Vidimo, da se izrazi počasi le kompliciajo. Če bi razpisali še vse funkcije, ki nastopajo v definiciji, bi dobili zelo dolg in nepregleden izraz. Vseeno smo pokazali, da se v lambda računu načeloma da izraziti tudi netrivialne algoritme. Na tem mestu končajmo s programiranjem v lambda računu.

Del diplomskega seminarja je tudi interpreter za lambda račun, kjer lahko preverimo, da so definirani izrazi res dobri. Navodila za uporabo so v dodatku C.

Dodatek A. Lastnosti α-ekvivalence

Definicija relacije $\equiv_\alpha$ je bila asimetrična, zato moramo dokazati, da je podana relacija res ekvivalenčna.

Tranzitivnost in refleksivnost relacije sta jasni iz definicije, za simetričnost pa se moramo malo bolj potruditi. Ni res, da lahko vsako α-pretvorbo obrnemo s samo eno α-pretvorbo. Poglejmo si npr. izraz $\lambda x.(\lambda y.y\, x)$. Po definiciji je $\lambda y.(\lambda y.y\, x)\, [x \to y] \equiv \lambda y.(\lambda z.z\, y)$, ampak iz tega izraza s samo eno uporabo α-pretvorbe ne moremo dobiti nazaj prvotnega izraza. Lahko ga dobimo z dvema α-pretvorbama, vendar dokazati, da lahko vedno eno α-pretvorbo obrnemo s končno mnogo α-pretvorbami zahteva nekaj dela.

V ta namen definirajmo še eno, navidez manjšo relacijo.

Definicija A.1. *Pravimo, da izraz P α_0-pretvorimo v izraz Q in pišemo $P \mapsto_{\alpha_0} Q$, če dobimo Q iz P tako, da zamenjamo nastop podizraza $\lambda x.M$ v izrazu P z izrazom $\lambda y.M\, [x \to y]$, kjer y v izrazu M* ne nastopa*, x pa ne nastopa vezano v M.*

Pišemo $P \rhd_{\alpha_0} Q$, če lahko dobimo Q iz P s končnim zaporedjem $\mapsto_{\alpha_0}$-pretvorb.

Pišemo $P \equiv_{\alpha_0} Q$, če lahko dobimo Q iz P s končnim zaporedjem α_0-pretvorb oz. obratnih α_0 pretvorb.

Razlika med to relacijo in prej definirano α-pretvorbo je, da sedaj zahtevamo, da y sploh ne nastopa v M, medtem ko smo prej zahtevali samo, da y ne nastopa prosto, poleg tega pa zahtevamo še, da x ne nastopa vezano v M. S tem se bomo izognili uporabi točk (4) in (7) definicije substitucije.

Lema A.2. *Če y in x nista vezana v M in y ni prost v M, velja:*

(1) *$M[x \to y]$ dobimo tako, da enostavno zamenjamo vse nastope spremenljivke x z y. Vsi ostali podizrazi ostanejo enaki,*
(2) *x ni prost v $M[x \to y]$ in nobena izmed x in y ni vezan $M[x \to y]$,*
(3) *$M[x \to y][y \to x] \equiv M$,*
(4) *$\mapsto_{\alpha_0}$ je simetrična relacija; velja $\lambda y.M[x \to y] \mapsto_{\alpha_0} \lambda x.M$,*
(5) *$M \equiv_{\alpha_0} N \Leftrightarrow M \rhd_{\alpha_0} N$.*

Dokaz. (1) Enostavno z indukcijo na strukturo izraza. Pogoji na x in y so postavljeni tako, da nikoli ne uporabimo točk (4) in (7) definicije substitucije.
(2) Sledi iz prve točke.
(3) Zopet z enostavno indukcijo na strukturo izraza. Pogoji na x in y, skupaj s točko (2) te leme, zagotavljajo, da nikoli ne uporabimo točke (7) definicije substitucije.
(4) Točka (2) te leme zagotavlja, da $M[x \to y]$ zadošča pogojem za α_0-pretvorbo. Velja torej $\lambda y.M[x \to y] \mapsto_{\alpha_0} \lambda x.M[x \to y][y \to x]$, kar pa je po točki (3) te leme enako $\lambda x.M$.
(5) Sledi iz prejšnje točke.

□

Dokazali smo, da lahko vse α_0-pretvorbe obrnemo. Poglejmo sedaj, kako nam to pomaga pri prej zastavljeni nalogi.

Lema A.3. *Naj bo M lambda izraz, x poljubna spremenljivka in y spremenljivka, ki ne nastopa prosto v M. Potem obstaja lambda izraz N, v katerem y sploh ne nastopa, x pa ni vezan v N, da velja*

$$M \rhd_{\alpha_0} N \text{ in } M[x \to y] \rhd_{\alpha_0} N[x \to y].$$

Dokaz. Z indukcijo na dolžino izraza.

M je spremenljivka: Vzamemo $N \equiv M$.
$M \equiv P\,Q$: Trditev sledi neposredno iz indukcijske predpostavke na P in Q, definicije substitucije in tranzitivnosti relacije $\rhd_{\alpha_0}$.
$M \equiv \lambda x.P$: Po indukcijski predpostavki obstaja lambda izraz Q, v katerem x ne nastopa vezano, y pa sploh ne, da veljajo zahtevane lastnosti. Potem velja $M \rhd_{\alpha_0} \lambda x.Q \rhd_{\alpha_0} \lambda z.Q[x \to z]$. Ker x v izrazu $\lambda z.Q[x \to z]$ po lemi 2.14 prosto ne nastopa, velja tudi druga zahtevana lastnost.

Podobno pokažemo še v ostalih primerih. □

Izrek A.4. *Relacije $\equiv_\alpha$, $\rhd_\alpha$ in $\equiv_{\alpha_0}$ so enake.*

Dokaz. Dokazati moramo, da lahko vsako α-pretvorbo izrazimo z zaporedjem α_0-pretvorb. Potem bo izrek sledil, saj že vemo, da α_0 pretvorbe lahko obrnemo in ker so očitno α_0-pretvorbe tudi α-pretvorbe, lahko α-pretvorbe obrnemo z α-pretvorbami.

Naj bo M lambda izraz. Če y ni prost v M po prejšnji lemi obstaja lambda izraz N, da velja $M \rhd_{\alpha_0} N$, kjer x ne nastopa vezano v N in y ne nastopa sploh v N.

Potem po definiciji velja tudi $\lambda x.M \rhd_{\alpha_0} \lambda x.N$. Uporabimo lahko $\mapsto_{\alpha_0}$-pretvorbo in dobimo

$$\lambda x.M \rhd_{\alpha_0} \lambda x.N \mapsto_{\alpha_0} \lambda y.N\,[x \to y] \rhd_{\alpha_0} \lambda y.M\,[x \to y]\,,$$

saj smo že pokazali, da je relacija $\rhd_{\alpha_0}$ simetrična. □

Dodatek B. Lambda račun je konsistenten

Pokazali bomo, da relacija $=_\beta$ ni prevelika, torej da obstajajo lambda izrazi, ki niso β-enaki. Posebej bomo pokazali, da za $n \neq m$ velja tudi $\overline{n} \neq_\beta \overline{m}$. Standarden dokaz tega dejstva v literaturi poteka preko prvega Church-Rosserjevega izreka, vendar je dokaz tega izreka predolg, da bi ga vključili v to delo, zato bomo to dokazali na drug način; s pomočjo Scottovega modela lambda računa. Lambda izraze bomo preslikali v elemente ustrezne množice in pokazali, da se β-enaka izraza slikata v isti element. Iz tega bo potem sledilo, da obstajajo β-različni izrazi. S tem bomo tudi upravičili predstavo, da lambda izrazi predstavljajo funkcije med ustreznimi množicami. Ta dokaz je iz [Bau10].

Težava pri preslikavi lambda izrazov v funkcije je, da so v lambda računu dovoljeni izrazi oblike $x\,x$ in zato potrebujemo prostor D, za katerega je podmnožica funkcij $D \to D$ na nek način vložena v D. Jasno je, da ne moremo vzeti kar vseh funkcij. Tukaj nastopijo enumeracije.

B.1. **Enumeracije.** Definirajmo funkcijo

$$\begin{aligned}\langle -, - \rangle : \mathbb{N}_0 \times \mathbb{N}_0 &\to \mathbb{N}_0 \\ : (m, n) &\mapsto 2^m(2n+1).\end{aligned}$$

Funkcija priredi vsakemu paru naravnih števil neko naravno število, *kodo* tega para. Iz teorije števil sledi, da je ta funkcija injektivna, ni pa surjektivna, saj 0 ni slika nobenega para.

Z iteracijo te funkcije lahko priredimo n-tericam naravnih števil ustrezne kode. Funkcijo označimo z $[\ldots]$ in jo definiramo rekurzivno

$$\begin{aligned}[\,] &= 0 \\ [n_1, n_2, \ldots, n_k] &= \langle n_1, [n_2, n_3, \ldots, n_k] \rangle.\end{aligned}$$

Ker je n-terica s kodo natanko določena v zapisu ne bomo ločevali med n-terico in njeno kodo. Za n-terico $s = [n_1, n_2, \ldots, n_k]$ definiramo še množico njenih elementov $\hat{s} = \{n_1, n_2, \ldots, n_k\}$.

Definicija B.1. *Za podmnožici $S, T \subseteq \mathbb{N}_0$ pravimo, da je S* daleč pod *T in pišemo $S \ll T$, če je S končna podmnožica množice T.*

Preslikava $f : \mathcal{P}(\mathbb{N}_0) \to \mathcal{P}(\mathbb{N}_0)$ je enumeracija*, če ima lastnost*

$$\forall S \in \mathcal{P}(\mathbb{N}_0) : f(S) = \bigcup_{A \ll S} f(A).$$

*Preslikava $g : \mathcal{P}(\mathbb{N}_0)^k \to \mathcal{P}(\mathbb{N}_0)$ je k-*tiška enumeracija*, če so zožitve na vsako podmnožico $\mathcal{P}(\mathbb{N}_0)$ enumeracije.*

Enumeracija je torej natanko določena z delovanjem na končnih podmnožicah in to bomo izkoristili.

Lema B.2. *Naj bo f enumeracija, $S \subseteq \mathbb{N}_0$ in B taka končna množica, da velja $B \ll \bigcup_{A \ll S} f(A)$. Potem obstaja množica $A \ll S$, da velja $B \subseteq f(A)$.*

Dokaz. Naj bo $B = \{b_1, b_2, \ldots, b_n\}$. Po predpostavki za $i \in \{1, 2, \ldots, n\}$ obstajajo množice $A_i \ll S$, da velja $b_i \in f(A_i)$. Označimo $A = \bigcup_{i=1}^{n} A_i$. Ker je f enumeracija velja $f(A) = \bigcup_{A' \ll A} f(A')$ in ker je $A_i \ll A$ za vsak i, velja $B \ll f(A)$. □

Lema B.3. *Kompozitum enumeracij je enumeracija.*

Dokaz. Naj bosta f in g enumeraciji

$$g(f(S)) = \bigcup_{B \ll f(S)} g(B) = \{x \mid \exists B \ll \bigcup_{A \ll S} f(A) : x \in g(B)\}$$

in ker je B končna in f enumeracija lahko uporabimo lemo B.2 ter dobimo

$$\begin{aligned} &= \{x \mid \exists A \ll S : \exists B \ll f(A) : x \in g(B)\} \\ &= \bigcup_{A \ll S} \bigcup_{B \ll f(A)} g(B) \\ &= \bigcup_{A \ll S} g(f(A)). \end{aligned}$$ □

Ugotovili smo že, da je enumeracija določena z delovanjem na končnih podmnožicah in zato lahko definiramo preslikavo, ki enumeraciji priredi podmnožico naravnih števil, s predpisom

$$\Gamma(f) = \{\langle s, n\rangle \in \mathbb{N}_0 \mid n \in f(\hat{s})\},$$

kjer je s seznam poljubne dolžine.

Obratno, vsaka podmnožica $S \subseteq \mathbb{N}_0$ določa enumeracijo $\Lambda(S)$, podano s predpisom

$$\Lambda(S)(T) = \{n \in \mathbb{N}_0 \mid \exists\, \hat{s} \subseteq T : \langle s, n\rangle \in S\},$$

kjer $\exists\, \hat{s} \subseteq T$ pomeni, da obstaja število s, ki je koda neke n-terice, katere elementi so elementi množice T.

Lema B.4. *Naj bo f enumeracija. Potem velja $\Lambda(\Gamma(f))(S) = f(S)$.*

Dokaz.

$$\Lambda(\Gamma(f))(S) = \{n \in \mathbb{N}_0 \mid \exists\, \hat{s} \subseteq S : \langle s, n\rangle \in \Gamma(f)\}$$

po definiciji Λ in če sedaj upoštevamo še definicijo Γ, dobimo

$$= \{n \in \mathbb{N}_0 \mid \exists \hat{s} \subseteq S : n \in f(\hat{s})\}$$

kar je po definiciji unije enako

$$= \bigcup_{\hat{s} \ll S} f(\hat{s})$$

kar je po predpostavki, da je f enumeracija, enako

$$= f(S),$$

saj je $\hat{s}$ lahko poljubna končna množica. □

Lema B.5. *Preslikava $(S, T) \mapsto \Lambda(S)(T)$ je enumeracija.*

Dokaz. Preveriti moramo po komponentah. Najprej prva.

$$\bigcup_{A \ll S} \Lambda(A)(T) = \bigcup_{A \ll S} \{n \in \mathbb{N}_0 \,|\, \exists \hat{s} \subseteq T : \langle s, n\rangle \in A\}$$

in če sedaj uporabimo definicijo unije, dobimo

$$= \{n \in \mathbb{N}_0 \,|\, \exists \hat{s} \subseteq T \, \exists A \ll S : \langle s, n\rangle \in A\},$$

kar pa je enako

$$= \{n \in \mathbb{N}_0 \,|\, \exists \hat{s} \subseteq T : \langle s, n\rangle \in S\} = \Lambda(S)(T),$$

saj lahko s končnimi množicami pokrijemo ves S.

Preverimo še, da je preslikava enumeracija tudi v drugi komponenti.

$$\bigcup_{A \ll T} \Lambda(S)(A) = \bigcup_{A \ll T} \{n \in \mathbb{N}_0 \,|\, \exists \hat{s} \subseteq A : \langle s, n\rangle \in S\}$$

in če sedaj uporabimo definicijo unije, dobimo

$$= \{n \in \mathbb{N}_0 \,|\, \exists A \ll T : \exists \hat{s} \subseteq A : \langle s, n\rangle \in S\},$$

kar pa je enako

$$= \{n \in \mathbb{N}_0 \,|\, \exists \hat{s} \subseteq T : \langle s, n\rangle \in S\} = \Lambda(S)(T),$$

po tranzitivnosti vsebovanosti in dejstva, da lahko T pokrijemo s končnimi množicami. □

Potrebujemo samo še eno dejstvo o enumeracijah.

Lema B.6. *Če je $f : \mathcal{P}(\mathbb{N}_0)^k \times \mathcal{P}(\mathbb{N}_0) \to \mathcal{P}(\mathbb{N}_0)$ enumeracija, je enumeracija tudi $g : \mathcal{P}(\mathbb{N}_0)^k \to \mathcal{P}(\mathbb{N}_0)$, definirana s predpisom*

$$g(\vec{S}) = \Gamma(U \mapsto f(\vec{S}, U)) = \{\langle s, n\rangle \in \mathbb{N}_0 \,|\, n \in f(\vec{S}, \hat{s})\}.$$

Dokaz. Naj bo $\vec{S}_{A_i} = (S_1, S_2, \ldots, S_{i-1}, A_i, S_{i+1}, \ldots, S_n)$

$$\begin{aligned}
\bigcup_{A_i \ll S_i} g(\vec{S}_{A_i}) &= \bigcup_{A_i \ll S_i} \{\langle s, n\rangle \in \mathbb{N}_0 \,|\, n \in f(\vec{S}_{A_i}, \hat{s})\} \\
&= \{\langle s, n\rangle \in \mathbb{N}_0 \,|\, \exists A_i \ll S_i : n \in f(\vec{S}_{A_i}, \hat{s})\} \\
&= \{\langle s, n\rangle \in \mathbb{N}_0 \,|\, n \in \bigcup_{A_i \ll S_i} f(\vec{S}_{A_i}, \hat{s})\} \\
&= \{\langle s, n\rangle \in \mathbb{N}_0 \,|n \in f(\vec{S}, \hat{s})\} = g(\vec{S}).
\end{aligned}$$

□

B.2. **Model $\mathcal{P}(\mathbb{N}_0)$.** Uporabimo sedaj prejšnje leme o enumeracijah in preslikajmo lambda izraze v enumeracije.

Definicija B.7. *Naj bo $\vec{x} = x_1, x_2, \ldots, x_n$ in M lambda izraz, za katerega velja $FV(M) \subseteq \vec{x}$. Potem induktivno na strukturo izraza M definiramo k-tiško enumeracijo*

$$[\![M]\!]_{\vec{x}} : \mathcal{P}(\mathbb{N}_0)^k \to \mathcal{P}(\mathbb{N}_0)$$

s predpisi:

$$\begin{aligned}
[\![x_j]\!]_{\vec{x}}(\vec{S}) &= S_k, \text{ kjer je } x_k \equiv x_j \text{ in } x_i \not\equiv x_j \text{ za } i > k \\
[\![P\,Q]\!]_{\vec{x}}(\vec{S}) &= \Lambda([\![P]\!]_{\vec{x}}(\vec{S}))([\![Q]\!]_{\vec{x}}(\vec{S})), \\
[\![\lambda y.P]\!]_{\vec{x}}(\vec{S}) &= \Gamma(T \mapsto [\![P]\!]_{\vec{x},y}(\vec{S}, T)).
\end{aligned}$$

$\vec{x}$ imenujemo kontekst*.*

Lahko je pokazati, da je projekcija res enumeracija, iz lem B.5 in B.6 ter B.3 pa sledi, da sta tudi predpisa za pretvorbo aplikacije in abstrakcije enumeraciji.

Poglejmo si primer.

Primer B.8.

$$\begin{aligned}
[\![\lambda x.x]\!]_{\vec{x}}(\vec{S}) &= \Gamma(T \mapsto [\![x]\!]_{\vec{x},x}(\vec{S}, T)) \\
&= \Gamma(T \mapsto T) \\
&= \{\langle s, n\rangle \in \mathbb{N}_0 \mid n \in \hat{s}\}.
\end{aligned}$$

Velja torej $[\![\lambda x.x]\!]_{\vec{x}}(\vec{S}) = \Gamma(\mathrm{id})$, kar je smiselno, saj si $\lambda x.x$ predstavljamo kot identiteto. V tem primeru lahko opazimo še nekaj več. Nikjer nismo uporabili dejstva, da je spremenljivka x res spremenljivka x. Enako bi dobili, če bi izračunali $[\![\lambda y.y]\!]_{\vec{x}}(\vec{S})$ za poljubno spremenljivko y. Opazimo lahko tudi, da kontekst $\vec{x}$ in množice $\vec{S}$ niso vplivale na končni izraz, saj spremenljivke $\vec{x}$ niso nastopale prosto v izrazu.

V nadaljevanju bomo implicitno predpostavili, da je kontekst dovolj velik, da je preslikava $[\![-]\!]_{\vec{x}}$ dobro definirana.

Lema B.9. *Naj bo M lambda izraz. Če x_i ne nastopa prosto v M, ali če obstaja $j > i$, da je $x_i \equiv x_j$, velja*

$$\forall S \in \mathcal{P}(\mathbb{N}_0)^n : [\![M]\!]_{\vec{x}}(\vec{S}) = [\![M]\!]_{\vec{y}}(\vec{T}),$$

kjer je $\vec{y} = x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ in $\vec{T} = S_1, S_2, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n$.

Dokaz. Z indukcijo na strukturo M. Če je M spremenljivka je trditev očitna. Prav tako sledi trditev enostavno po indukcijski predpostavki, če je M aplikacija. Poglejmo si primer, ko je M abstrakcija. Ločimo dva primera:

(1)

$$[\![M \equiv \lambda x_i.N]\!]_{\vec{x}}(\vec{S}) = \Gamma(U \mapsto [\![N]\!]_{\vec{x},x_i}(\vec{S}, U))$$

kar je po indukcijski predpostavki za N enako

$$= \Gamma(U \mapsto [\![N]\!]_{\vec{y},x_i}(\vec{T}, U)),$$

saj dodamo x_i za spremenljivko x_i v $\vec{S}$, kar pa je po definiciji enako

$$= [\![\lambda x_i.N]\!]_{\vec{y}}(\vec{T}).$$

(2)

$$[\![M \equiv \lambda z.N]\!]_{\vec{x}}(\vec{S}) = \Gamma(U \mapsto [\![N]\!]_{\vec{x},z}(\vec{S}, U))$$

kar je po indukcijski predpostavki za N enako

$$= \Gamma(U \mapsto [\![N]\!]_{\vec{y},z}(\vec{T}, U)),$$

saj x_i ne nastopa prosto v N, kar pa je po definiciji enako

$$= [\![\lambda z.N]\!]_{\vec{y}}(\vec{T}). \qquad \square$$

Lema B.10. *Naj bo M lambda izraz v katerem y ne nastopa. Potem velja*

$$\forall S \in \mathcal{P}(\mathbb{N}_0)^n : [\![M]\!]_{\vec{x}}(\vec{S}) = [\![M\,[x \to y]]\!]_{\vec{y}}(\vec{S}),$$

kjer je $\vec{y} = x_1, x_2, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n$, $\vec{x} = x_1, x_2, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n$ in $x_j \not\equiv y$ ter $x_j \not\equiv x$ za $j > i$.

Dokaz. Če je M spremenljivka trditev drži po definiciji preslikave $[\![.]\!]_{\vec{x}}$. Prav tako trditev enostavno iz indukcijske predpostavke, če je M aplikacija. Poglejmo si primer, ko je M abstrakcija. Zopet ločimo dva primera:

- Če je $M \equiv \lambda x.N$ trditev sledi po lemi B.9, saj niti x niti y nista prosta v M.
- Naj bo sedaj $M \equiv \lambda z.N$.

$$[\![\lambda z.N]\!]_{\vec{x}}(\vec{S}) = \Gamma(T \mapsto [\![N]\!]_{\vec{x},z})(\vec{S}, T)$$

kar je po indukcijski predpostavki za N enako

$$\begin{aligned} &= \Gamma(T \mapsto [\![N\,[x \to y]]\!]_{\vec{y},z}(\vec{S}, T)) \\ &= [\![\lambda z.N\,[x \to y]]\!]_{\vec{y}}(\vec{S}) \\ &= [\![(\lambda z.N)\,[x \to y]]\!]_{\vec{y}}(\vec{S}). \end{aligned} \qquad \square$$

Posledica te leme je zelo pomembna.

Posledica B.11. *Naj bo $\lambda x.M$ lambda izraz in naj y ne nastopa v M. Potem velja*

$$\forall S \in \mathcal{P}(\mathbb{N}_0)^n : [\![\lambda x.M]\!]_{\vec{x}}(\vec{S}) = [\![\lambda y.M\,[x \to y]]\!]_{\vec{x}}(\vec{S}).$$

Dokaz. Posledica sledi iz definicije 2.1 in prejšnje leme. $\square$

Po izreku A.4, lemi A.2 ter definiciji relacije $\rhd_{\alpha_0}$ sedaj sledi naslednji izrek.

Izrek B.12. *Za poljubna lambda izraza M in N velja:*

$$M \equiv_\alpha N \Rightarrow [\![M]\!]_{\vec{x}}(\vec{S}) = [\![N]\!]_{\vec{x}}(\vec{S}).$$

Tehnično podrobnost smo odpravili. Pokažimo sedaj še, da se tudi β-enaki izrazi s preslikavo $[\![.]\!]$ slikajo v iste točke.

Lema B.13. *Naj bo M lambda izraz. Naj bo $\vec{x}$ seznam spremenljivk in naj $\vec{y}$ označuje seznam spremenljivk, kjer sta i-ta in $i+1$-va spremenljivka zamenjani. Podobno naj $\vec{T}$ označuje seznam množic, kjer sta i-ta in $i+1$-va množica zamenjani. Potem velja*

$$[\![M]\!]_{\vec{x}}(\vec{S}) = [\![M]\!]_{\vec{y}}(\vec{T}).$$

Dokaz znova poteka z indukcijo na strukturo izraza ob uporabi leme B.9.

Izrek B.14. *Za poljubna lambda izraza M in N velja:*

$$[\![(\lambda x.M)\, N]\!]_{\vec{x}}(\vec{S}) = [\![M\,[x \to N]]\!]_{\vec{x}}(\vec{S}).$$

Dokaz. Po izreku B.12 lahko brez škode za splošnost predpostavimo, da nobena spremenljivka vezana v M ni prosta v N, ter da spremenljivka x ni vezana v M. Dobimo

$$[\![(\lambda x.M)\, N]\!]_{\vec{x}}(\vec{S}) = \Lambda(\Gamma(T \mapsto [\![M]\!]_{\vec{x},x}(\vec{S},T)))([\![N]\!]_{\vec{x}}(\vec{S})),$$

kar je po lastnosti izrazov Λ in Γ enako

$$= (T \mapsto [\![M]\!]_{\vec{x},x}(\vec{S},T))([\![N]\!]_{\vec{x}}(\vec{S})) = [\![M]\!]_{\vec{x},x}(\vec{S},[\![N]\!]_{\vec{x}}(\vec{S})).$$

Pokažimo sedaj še, da je tudi desna stran enaka dobljenemu izrazu. Dokazujemo z indukcijo na strukturo izraza M. Če je M spremenljivka dobimo enakost takoj po definiciji. Če je M aplikacija zopet dobimo enakost trivialno po indukcijski predpostavki in definiciji substitucije na aplikaciji. Poglejmo si primer, ko je M abstrakcija. Premislili smo že, da lahko predpostavimo, da x ni vezan v M in da nobena spremenljivka vezana v M ni prosta v N.

$$\begin{aligned}[\![(\lambda z.P)\,[x \to N]]\!]_{\vec{x}}(\vec{S}) &= [\![\lambda z.P\,[x \to N]]\!]_{\vec{x}}(\vec{S}),\\ &= \Gamma(T \mapsto [\![P\,[x \to N]]\!]_{\vec{x},z}(\vec{S},T))\end{aligned}$$

kar je po indukcijski predpostavki za P enako

$$= \Gamma(T \mapsto [\![P]\!]_{\vec{x},z,x}(\vec{S},T,[\![N]\!]_{\vec{x},z}(\vec{S},T)))$$

in če uporabimo lemo B.9, upoštevajoč predpostavko, da z ne nastopa prosto v N, dobimo

$$= \Gamma(T \mapsto [\![P]\!]_{\vec{x},z,x}(\vec{S},T,[\![N]\!]_{\vec{x}}(\vec{S})))$$

in če sedaj upoštevamo še lemo B.13, dobimo

$$= \Gamma(T \mapsto [\![P]\!]_{\vec{x},x,z}(\vec{S},[\![N]\!]_{\vec{x}}(\vec{S}),T)),$$

kar pa je po definiciji enako

$$= [\![\lambda z.P]\!]_{\vec{x},x}(\vec{S},[\![N]\!]_{\vec{x}}(\vec{S})). \qquad \square$$

Iz tega izreka ter izreka B.12 po indukciji na število pretvorb sledi, da če sta izraza β-enaka, sta tudi njuni sliki enaki.

B.3. **Lambda račun je konsistenten.** Pokazali bomo, da izraza $\mathbf{T}$ in $\mathbf{F}$ nista β-enaka. Velja

$$[\![\mathbf{T}]\!]_{\vec{x}}(\vec{S}) = \Gamma(T \mapsto (\Gamma(U \mapsto T))$$

in

$$[\![\mathbf{F}]\!]_{\vec{x}}(\vec{S}) = \Gamma(T \mapsto (\Gamma(U \mapsto U)).$$

Denimo sedaj, da velja $[\![\mathbf{T}]\!]_{\vec{x}}(\vec{S}) = [\![\mathbf{F}]\!]_{\vec{x}}(\vec{S})$. Potem velja

$$\begin{aligned}A = [\![x]\!]_{\vec{x},x,y}(\vec{S},A,B) &= \Lambda(\Lambda([\![\mathbf{T}]\!]_{\vec{x}}(\vec{S}))(A))(B)\\ &= \Lambda(\Lambda([\![\mathbf{F}]\!]_{\vec{x}}(\vec{S}))(A))(B)\\ &= [\![y]\!]_{\vec{x},x,y}(\vec{S},A,B) = B\end{aligned}$$

za poljubni množici A in B. To je očitno nesmisel, saj $\mathcal{P}(\mathbb{N}_0)$ vsebuje vsaj dve različni množici.

Sedaj lahko pokažemo še, da so Scottovi numerali različne β-normalne oblike. S pomočjo izraza $\mathbf{D}$ lahko hitro pokažemo, da je $\overline{0} \neq_\beta \overline{k}$ za $k \geq 1$. Res, $\mathbf{D}\overline{0} =_\beta \mathbf{T}$ in $\mathbf{D}\overline{k} =_\beta \mathbf{F}$. Če bi bil $\overline{0} =_\beta \overline{k}$, bi veljalo $\mathbf{D}\overline{k} =_\beta \mathbf{T}$ in zato $\mathbf{F} =_\beta \mathbf{T}$, kar smo pokazali, da ne velja.

Od tod sledi, da za $m \neq n$ velja $\overline{m} \neq_\beta \overline{n}$. Če bi veljalo $\overline{m} =_\beta \overline{n}$ (brez škode za splošnost lahko vzamemo, da je $n > m$), potem bi veljalo $\mathbf{P}\,\overline{m} =_\beta \overline{m-1} =_\beta \mathbf{P}\,\overline{n} =_\beta \overline{n-1}, \cdots, \overline{n-m} =_\beta \overline{0}$, kar pa smo pokazali, da ne velja.

Dodatek C. Interpreter

Del seminarja je tudi interpreter za lambda račun. V osnovi lahko vpisujemo lambda izraze, interpreter pa jih poskusi poenostaviti do normalne oblike. Če izraz nima normalne oblike moramo prekiniti izvajanje, saj v splošnem ni mogoče ugotoviti, ali nek izraz ima normalno obliko ali ne.

Izraze vpisujemo kot v predhodnih razdelkih, le da namesto grške črke λ pišemo nagibnico, `\`. Spremenljivke so lahko sestavljene iz več znakov, a morajo biti ločene s presledki. Vnos izraza zaključimo s tipko Enter. Poglejmo si primer.

```
> (\x.x)(\y.y)
\y.y
```

`>` na začetku vrstice pomeni vnos uporabnika. V tem primeru smo vnesli lambda izraz $(\lambda x.x)\,(\lambda y.y)$. Normalna oblika tega izraza je $\lambda y.y$.

S ključno besedo `let` lahko definiramo začasne spremenljivke. Kasneje lahko spremenljivko uporabimo pri vnosu, interpreter pa jih upošteva tudi pri izpisu, da so izrazi bolj pregledni. Poseben primer so Scottovi numerali, ki jih interpreter vedno izpiše kot števila. Definirajmo števila 0, 1 in 2 ter funkcijo naslednik.

```
> let 0 = \x y.x
> let 1 = \x y.y (\x y.x)
> let 2 = \x y.y (\x y.y (\x y.x))
> let N = \n x y.y n
> N 0
1
> N 1
2
> N 2
3
```

Uporabimo `N` in definirajmo še seštevanje. Oznake so enake kot v prejšnjih razdelkih.

```
> let P = \n.n 0 (\x.x)
> let Y = \f.(\x.f (x x)) (\x.f (x x))
> let T = \x y.x
> let F = \x y.y
> let D = \x.x T (\y.F)
> let R = Y (\x y1 y2 y3.(D y1 y2 (y3 (P y1) (x (P y1) y2 y3))))
```

```
> let plus = \y x.R y x (\u v.N v)
> plus 1 2
3
> let 3 = plus 1 2
> plus 2 3
5
```

Izraze lahko zapišemo tudi v datoteko, ki jo potem naložimo v interpreter z ukazom `:load ime_datoteke`. Interpreter bere vrstico po vrstico in izvršuje ukaze. Seznam vseh ukazov s kratkimi opisi lahko dobimo z ukazom `:help`, interpreter pa zapremo z ukazom `:exit`.

V datoteki `test.lambda` je že definirana večina izrazov, ki so uporabljeni v tem delu.

Literatura

[Bar84] Hendrik Pieter Barendregt, *The Lambda Calculus – Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics, vol. 103, North-Holland, 1984.

[Bau10] Andrej Bauer, *Konsistentnost lambda računa*, Seminar za osnove matematike in teoretičnega računalništva, April 2010, Inštitut za matematiko in fiziko.

[CF58] Haskell B. Curry and Robert Feys, *Combinatory logic*, vol. 1, North Holland, 1958, Second edition, 1968.

[Hin08] Jonathan P. Hindley, J. Roger in Seldin, *Lambda-Calculus and Combinators: An Introduction*, Cambridge University Press, New York, NY, USA, 2008.

[Lan65] P. J. Landin, *Correspondence between ALGOL 60 and Church's Lambda-notation: part I*, Commun. ACM **8** (1965), no. 2, 89–101.

[Pri94] Niko Prijatelj, *Osnove matematične logike 3*, DMFA, Ljubljana, 1994.

[Tur37] A. M. Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proc. London Math. Soc. **s2-42** (1937), no. 1, 230–265.