Poglavje 6

Λ-RAČUN

6.1 Uvod

Lambda račun (λ-račun) je razvil leta 1930 Alonzo Church. Uporabljen je bil leta 1936 kot osnovni jezik v slavnem članku o obstoju neodločljivega problema (Entscheidungsproblem).

Λ-račun lahko vidimo kot *primalen* programski jezik, ki je izrazno enako močen kot Turingov stroj. Alan Turing je v istem času (okoli leta 1936) podobno kot Church pokazal, da ni mogoče rešiti vse probleme izražene v univerzalnem jeziku in zato obstajajo *neodločljivi* problemi. Turing je dokazal tudi ekvivalentnost med svojim strojem in λ-računom.

Iz strani matematike predstavlja λ-račun formalno osnovo s pomočjo katere je mogoče izražati *funkcije*. Iz strani računalništva predstavlja λ-račun formalen jezik, ki je osnova vsem programskim jezikom[1]. Začetna verzija programskega jezika Lisp, na primer, je genialno enostavna implementacija λ-računa.

Λ-račun je v zadnji polovici stoletja uporabljen tudi kot formalna paradigma programskih jezikov. V zadnjih desetletjih, ko se je razširilo raziskovalno delo na *teoriji programskih jezikov* je λ-račun uporabljen kot osnovno formalno orodje za definicijo *semantike* programskih jezikov.

Izkazalo se je, da lahko λ-račun služi kot osnova za opis pomena tudi bolj kompleksnih gradnikov programskih jezikov kot je npr. polimorfizem ter tudi gradnikov bolj kompleksnih programskih jezikov kot so npr. objektni programski jeziki.

V nadaljevanju bo najprej predstavljena sintaksa in evaluacija λ-računa. Sledila bo predstavitev postopkov redukcije λ-izrazov v vrednosti. Prikazana bo deterministič nost jezika in izrazna moč λ-računa.

[1] Peter J. Landin, ki je veliko prispeval na področju programskih jezikov je leta 1966 rekel: "Whatever the next 700 languages turn out to be, they will surely be variants of lambda calculus."

6.2 Sintaksa

Definicija 6.2.1. *Množica λ-izrazov Λ je zgrajena iz neskončne množice spremenljivk $V = \{v, v', v'', \ldots\}$ z aplikacijo in funkcijsko abstrakcijo.*

$$\begin{array}{rcl} x \in V & \Rightarrow & x \in \Lambda \\ M, N \in \Lambda & \Rightarrow & (M\ N) \in \Lambda \\ M \in \Lambda, x \in V & \Rightarrow & (\lambda x.M) \in \Lambda \end{array}$$

Izraz $\lambda x.M$ predstavlja λ-abstrakcijo s katero opišemo funkcije z enim argumentom in telesom M. Izraz $(M\ N)$ predstavlja aplikacijo funkcijskega izraza M na parametru N.

Backus-Naurjeva oblika zapisa sintakse je sledeča. Simboli e, e_1, e_2 predstavljajo λ-izraze.

$$e ::= v \mid \lambda x.e \mid e_1 e_2$$

Opazka 6.2.1. *Poglejmo si kako se povezujejo izrazi zgrajeni z aplikacijo funkcije in lambda abstrakcijo. Aplikacija funkcije je levo asociativna.*

$$x\ y\ z \equiv ((x\ y)z)$$

Lambda abstrakcija se povezuje na desno.

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

Primer 6.2.1. *Poglejmo si nekaj primerov λ-izrazov.*

$$\begin{array}{l} y \\ y\ x \\ \lambda x.y\ x \\ (\lambda x.y\ x)\ z \\ (\lambda x.\lambda y.y\ x)\ z\ w \end{array}$$

□

V vseh jezikih je pomembno *področje definicije spremenljivk* t.j. del programa kjer je definirana neka spremenljivka.

Abstrakcija $\lambda x.E$ povezuje spremenljivko x v E. Spremenljivka x je definirana znotraj izraza E ali E je področje definicije spremenljivke x. Pravimo, da je x *vezana v* E podobno kot so argumenti funkcije vezani na telo funkcije.

6.2.1 Proste in vezane spremenljivke

Spremenljivka x je prosta v izrazu E, če ima pojavitve, ki niso vezane z λ-abstrakcijo.
.

Definicija 6.2.2. *Množica* prostih spremenljivk *izraza M je definirana rekurzivno z naslednjimi pravili glede na strukturo izraza.*

$$\begin{array}{rcl} FV(x) & = & \{x\} \\ FV(E_1\ E_2) & = & FV(E_1) \cup FV(E_2) \\ FV(\lambda x.E) & = & FV(E) - \{x\} \end{array}$$

Primer 6.2.2. $FV(\lambda x.x\ (\lambda y.x\ y\ z)) = \{z\}$

Definicija 6.2.3. *Izraz M je zaprt λ-izraz, če velja $FV(M) = \emptyset$.*

Naslednji primeri prikažejo povezovanje λ-izrazov in proste spremenljivke.

Primer 6.2.3. *Naslednji λ-izraz vsebuje dve λ-abstrakciji in eno prosto spremenljivko.*

$$\begin{array}{rl} & \lambda x.\lambda y.x\ z\ y \\ = & \lambda x.(\lambda y.((x\ z)\ y)) \end{array}$$

Naslednji λ-izraz porazedeli parameter z po dveh funkcijah x in y. Ni prostih spremenljivk oz. izraz je zaprt.

$$\begin{array}{rl} & \lambda x.\lambda y.\lambda z.(x\ z)\ (y\ z) \\ = & \lambda x.(\lambda y.(\lambda z.((x\ z)\ (y\ z)))) \end{array}$$

Kot bomo videli kasneje predstavlja naslednji λ-izraz seštevanje naravnih števil. Tudi v tem primeru je λ izraz zaprt.

$$\begin{array}{rl} & \lambda m.\lambda n.\lambda z.\lambda s.m\ (n\ z\ s)\ s \\ = & \lambda m.(\lambda n.(\lambda z.(\lambda s.((m\ ((n\ z)\ s))\ s)))) \end{array}$$

□

6.3 Evaluacija

Lambda račun je zelo izrazen jezik, ki je enakovreden Turingovem stroju. V tej sekciji si bomo ogledali evaluacijo λ-računa.

Osnovna metoda uporabljena pri evaluacji λ-računa je *substitucija*: v sekvenci simbolov zamenjamo izbran simbol s sekvenco simbolov. Presentljivo je, da je mogoče na takšen način zelo enostavno realizirati jezik, ki služi kot formalna osnova programskih jezikov.

Osnovni dve "operaciji" λ-računa sta definicija funkcije z λ-abstrakcijo in aplikacija funkcije na parametru. Evaluacijo λ-izrazov bomo definirali na osnovi nekaj enostavnih pravil operacijske semantike.

Evaluacijo λ-izraza velikokrat imenujemo *redukcija*. λ-izraze reduciramo z uporabo dveh osnovnih pravil: *alfa konverzijo* in *beta redukcijo*

Poglejmo si najprej formalno definicijo substitucije, ki jo bomo uporabljali kasneje pri definiciji pravil za evaluacijo λ-izrazov.

6.3.1 Substitucija

Definicija 6.3.1. *Naj bo $M, N \in \Lambda$ ter $x \in V$. Operacijo substitucije M za x v N zapišemo kot $[N/x]M$. Substitucija je definirana s sledečimi pravili:*

$$[N/x]x = N$$
$$[N/x]z = z \Leftarrow z \neq x$$
$$[N/x](L\ M) = ([N/x]L)([N/x]M)$$
$$[N/x](\lambda z.M) = \lambda z.([N/x]M) \Leftarrow z \neq x \wedge z \notin FV(N)$$

Substitucijo uporabljamo za preimenovanje vezanih spremenljivk v pod-izrazu zato, da so unikatne znotraj celotnega izraza. To operacijo imenujemo tudi α-konverzija.

Druga uporaba substitucije je aplikacija λ-abstrakcije na parametru, ki jo izvedemo z tekstovno zamenjavo spremenljivke v λ-abstrakciji s parametrom. To operacijo imenujemo β-redukcija.

Primer 6.3.1. *Dan je izraz $\lambda y.(\lambda x.x)y\ x$ kjer želimo zamenjati spremenljivko x z izrazom $y\ (\lambda x.x)$.*

$$[y\ (\lambda x.x)/x]\lambda y.(\lambda x.x)\ y\ x$$

Najprej ugotovimo, da ima izraz $\lambda y.(\lambda x.x)y\ x$ dve različni spremenljivki x: prva je vezana v izrazu $(\lambda x.x)$, druga spremenljivka x pa je prosta. Zamenjati je potrebno imena spremenljivk. Običajno jih zamenjamo v zaprtih izrazih.

$$[y\ (\lambda x.x)/x]\lambda y.(\lambda u.u)\ y\ x$$

Koristno je enolično imenovati tudi spremenljivke v izrazih s katerimi je definirana substitucija. Spremenljivki x in y zamenjamo s spremenljivkama v in z.

$$[z\ (\lambda v.v)/x]\lambda y.(\lambda u.u)\ y\ x$$

Zdaj lahko nedvoumno izvedemo substitucijo x z $z\ (\lambda v.v)$.

$$[z\ (\lambda v.v)/x]\lambda y.(\lambda u.u)\ y\ x \equiv$$
$$\lambda y.(\lambda u.u)\ y\ z\ (\lambda v.v)$$

□

6.3.2 Redukcija

λ-izrazi, ki jih lahko pretvorimo med sabo s primenovanjem vezanih spremenljivk so identični.

Primer 6.3.2. $\lambda x.x \equiv \lambda y.y$

Definicija 6.3.2. *Dan je λ-izraz $\lambda x.M$. Preimenovanje vezane spremenljivke x v y imenujemo α-*konverzija*.*

$$\lambda x.M \equiv \lambda y.([y/x]M) \Leftarrow y \notin FV(M) \tag{6.1}$$

V osnovnem lambda računu je edini način za ovrednotenje izrazov aplikacija funkcij na argumentih.

Definicija 6.3.3. *Dan je λ-izraz $\lambda x.M$. Aplikacijo funkcije $(\lambda x.M)$ na argumentu N implementiramo z β-*redukcijo*:*

$$(\lambda x.M)\ N \rightarrow [N/x]M \tag{6.2}$$

Izraz $(\lambda x.M)$ imenujemo *redeks* (iz angl. reducable expression).

β-redukcijo λ-izraza M v N zapišemo $M \rightarrow_\beta N$ ali $M \rightarrow N$.

Definicija 6.3.4. *β-izpeljava λ-izraza je sestavljena iz večih zaporednih β-redukcij. β-izpeljavo M v N zapišemo:*

$$M \rightarrow_\beta^* N$$

Primer 6.3.3.

$$(\lambda x.x\ y)(u\ v) \rightarrow_\beta u\ v\ y$$
$$(\lambda x.\lambda y.x)z\ w \rightarrow_\beta (\lambda y.z)w \rightarrow z$$
$$(\lambda x.\lambda y.x)z\ w \rightarrow_\beta^* z$$

Definicija 6.3.5. *Λ-izraz M je β-*konvertibilen *z N ali $M =_\beta N$, če velja $M = N$, ali $\exists B : M \to^* B \wedge N \to^* B$.*

Primer 6.3.4.

$$(\lambda x.x)z =_\beta (\lambda x.\lambda y.x)z\ z \Longleftarrow \\ (\lambda x.\lambda y.x)z\ w \to^* z \wedge \\ (\lambda x.x)z \to^* z.$$

Sledi nekaj zanimivih primerov evaluacije λ-izrazov.

Primer 6.3.5. *Prvi primer predstavlja funkcijo identitete.*

$$(\lambda x.x)E \to [E/x]x = E$$

Naslednji primer tudi uporablja funkcijo identitete.

$$(\lambda f.f(\lambda x.x))(\lambda x.x) \to \\ [(\lambda x.x)/f]f(\lambda x.x) = [(\lambda x.x)/f]f(\lambda y.y) \to \\ (\lambda x.x)(\lambda y.y) \to \\ [(\lambda y.y)/x]x = \lambda y.y$$

Naslednji λ izraz se nikoli ne zaključi. Kot bo predstavljeno kasneje je možno na osnovi naslednjega izraza realizirati rekurzijo.

$$(\lambda x.xx)(\lambda y.yy) \to \\ [(\lambda y.yy)/x]xx = (\lambda x.xx)(\lambda y.yy) \to ...$$

□

6.3.3 Kombinatorji

Kombinatorji so osnovni primitivi λ-računa s katerimi lahko izrazimo nekatere poznane funkcije kot je npr. funkcija identitete, vejitvene stavke, in druge. S kombinatorji lahko izrazimo bolj abstraktne programe v λ-računu.

Prvi kombinator je funkcija identitete, ki jo označimo z I.

$$I = \lambda x.x$$

Funkciji K in K_* sprejmeta dva argumenta in zavržeta enega izmed argumentov. Funkciji sta uporabljeni za predstavitev boolovih vrednosti $true$ in $false$. V naslednji sekciji bo predstavljen primer uporabe K oz. K_*.

$$K = \lambda x.(\lambda y.x) \\ K_* = \lambda x.(\lambda y.y)$$

Naslednji kombinator S porazdeli zadnji argument kombinatorja kot parameter prvih dveh argumentov S. Argument funkcije S porazdelimo po dveh funkcijah.

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

Naslednji kombinator Ω je osnova za implementacijo rekurzije z λ-računom. V osnovni obliki Ω predstavlja funkcijo, ki se nikoli ne zaključi.

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

Rekurzivne funkcije lahko definiramo na osnovi kombinatorja Y, ki je osnovan na Ω. Funkcija Y ima za argument funkcijo f, ki jo želimo izvajati rekurzivno.

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

Najpomembnejša lastnost kombinatorja Y je izrek o fiksni točki, ki je bil predstavljen v prejšnjem poglavju. Izrek v okviru uporabe funkcije Y lahko zapišemo na sledeč način.

Izrek 6.3.1 (Izrek o fiksni točki)**.**

$$F(Y\ F) =_\beta Y\ F \tag{6.3}$$

Dokaz.

$$\begin{array}{rcl} Y\ F & = & \lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))\ F \\ & \rightarrow & (\lambda x.F(x\ x))(\lambda x.F(x\ x)) \\ & \rightarrow & F((\lambda x.F(x\ x))(\lambda x.F(x\ x))) \\ & \leftarrow & F((\lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x)))\ F) \\ & = & F\ (Y\ F) \end{array}$$

□

6.4 Programiranje v λ-računu

6.4.1 Curry

Lambda račun nima funkcij z večimi argumenti vendar je enostavno doseči isti učinek s *funkcijami višjega reda.*

Naj bo M izraz s prostima spremenljivkama x in y. Želimo napisati funkcijo F, ki za vsak par (N, L) zamenja x z N in y z L v izrazu M. V osnovnem lambda računu ne moremo napisati $F = \lambda(x, y).M$.

$F = \lambda x.\lambda y.M$

- F je funkcija, ki ob danem argumentu N vrne funkcijo, ki ob dani vrednosti za y vrne željeni rezultat.
- $FNL \rightarrow (\lambda y.[N/x]M)L \rightarrow [L/y][N/x]M$

Transformacijo, ki funkcijo z večimi argumenti prevede v funkcijo višjega reda imenujemo *Curry*

6.4.2 Logične vrednosti

Kako uvedemo logične vrednosti $\{True, False\}$ v Lambda račun.

Logične vrednosti so funkcije $True,\ False$

$$True = \lambda t.\lambda f.t$$
$$False = \lambda t.\lambda f.f$$

if stavek: if $= \lambda l.\lambda m.\lambda n.\ l\ m\ n$

Primer izpeljave (redukcije) if stavka:

if $True\ M\ N$	$=$	$(\lambda l.\lambda m.\lambda n.\ l\ m\ n)\ True\ M\ N$	po definiciji
	$\rightarrow$	$(\lambda m.\lambda n.\ True\ m\ n)\ M\ N$	β-redukcija
	$\rightarrow$	$(\lambda n.\ True\ M\ n)\ M$	β-redukcija
	$\rightarrow$	$True\ M\ N$	β-redukcija
	$=$	$(\lambda t.\lambda f.t)\ M\ N$	po definiciji
	$\rightarrow$	$(\lambda f.M)\ N$	β-redukcija
	$\rightarrow$	M	β-redukcija

Logični IN

Logični IN lahko zapišemo v naslednji obliki:

$$And = \lambda b.\lambda c.\ b\ c\ False$$

V primeru, da je b resnično potem vrne c sicer $False$.
Izraz c vrne $True$ samo, če je izraz resničen...

Primer aplikacije funkcije And: $And\ True\ False$

6.4.3 Churchova števila

$C_0 = \lambda z.\lambda s.z$
$C_1 = \lambda z.\lambda s.s\ z$
$C_2 = \lambda z.\lambda s.s(s\ z)$
$\vdots$
$C_n = \lambda z.\lambda s.s(s(\ldots(s\ z)\ldots)$

Število n je predstavljeno s funkcijo C_n, ki ima dva argumenta z (zero) in s (successor) ter aplicira n kopij funkcije s na z.

Število n je predstavljeno z n-kratno aplikacijo funkcije s na z.

Običajne aritmetične operacije na Churchovih številih so:

$Plus = \lambda m.\lambda n.\lambda z.\lambda s.m\ (n\ z\ s)\ s$
$Times = \lambda m.\lambda n.m\ \ C_0\ (Plus\ n)$

$(Plus\ 1\ 2) \rightarrow^* 3$
$Plus\ (\lambda z.\lambda s.s\ z)\ (\lambda z.\lambda s.s(s\ z)) \rightarrow$
$(\lambda m.\lambda n.\lambda z.\lambda s.m(n\ z\ s)s)\ (\lambda z.\lambda s.s\ z)\ (\lambda z.\lambda s.s(s\ z)) \rightarrow$
$(\lambda n.\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)(n\ z\ s)s)\ (\lambda z.\lambda s.s(s\ z)) \rightarrow$
$\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)((\lambda z.\lambda s.s(s\ z))\ z\ s)s \rightarrow$
$\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)((\lambda s.s(s\ z))\ s)s \rightarrow$
$\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)(s(s\ z))s =$
$\lambda z.\lambda s.(((\lambda z.\lambda s.s\ z)\ (s(s\ z)))s) \rightarrow$
$\lambda z.\lambda s.((\lambda s.s(s\ (s\ z)))s) \rightarrow$
$\lambda z.\lambda s.s(s\ (s\ z))$

Seštevanje ni preveč zabavno :(

6.4.4 Fakulteta

Intuitivna definicija funkcije, ki izračuna fakulteto danega števila.

if $n = 0$ then 1
else $n *$ (if $n - 1 = 0$ then 1
else $(n - 1) *$ (if $n - 2 = 0$ then1
else $(n - 2) * \ldots$

Rekurzijo lahko izrazimo s funkcijo $G = \lambda f.\langle telo\ f\rangle$ in $F = Y\ G$

$$\begin{array}{lll} F & = & Y\ G \\ & =_\beta & G\ (Y\ G) \\ & =_\beta & \langle telo\ (Y\ G)\rangle \\ & =_\beta & \langle telo\ \langle telo\ (Y\ G)\rangle\rangle \\ & & \vdots \end{array}$$

Funkcijo $Factorial$ lahko zdaj definiramo na sledeč način.

$$\begin{array}{lll} Fact & = & \lambda fact.\lambda n.\text{if}\ (IsZero\ n)\ C_1\ (Times\ n\ (fact(Pred\ n))) \\ Factorial & = & Y\ Fact \end{array}$$

Poglejmo si izpeljavo fakultete za C_2.

$Factorial\ C_2 = Y\ Fact\ C_2$
$=_\beta\ Fact\ (Y\ Fact)\ C_2$
$=_\beta\ (\lambda fact.\lambda n.\text{if}(IsZero\ n)\ C_1\ (Times\ n(fact(Pred\ n))))\ (Y\ Fact)\ C_2$
$=_\beta\ (\lambda n.\text{if}(IsZero\ n)\ C_1\ (Times\ n(Y\ Fact\ (Pred\ n))))\ C_2$

$=_\beta$ if$(IsZero\, C_2)\, C_1\, (Times\, C_2(Y\ Fact\, (Pred\, C_2)))$
$=_\beta$ if $False\, C_1\, (Times\, C_2(Y\ Fact\, C_1)))$
$=_\beta Times\, C_2(Y\ Fact\, C_1)$
$= Times\, C_2(Factorial\, C_1)$

6.5 Redukcija λ-izrazov

6.5.1 Operacijska semantika

V čisti obliki je λ-račun ne vsebuje drugega kot spremenljivke, λ abstrakcijo in aplikacijo. Edini način za evaluacijo λ-izrazov je redukcija redeksa, ki predstavlja aplikacijo funkcije na argumentu.

$$(\lambda x.M)N \rightarrow [N/x]M$$

Redukcijo λ-izraza realiziramo torej s prepisovanjem telesa λ-abstrakcije M s uporabo substitucije $[N/x]M$.

Redeks λ-izraza se lahko nahaja zunaj ali pa je vgnezden kot pod-izraz. V λ-izrazu imamo lahko več redeksov.

Poglejmo si zdaj pravila za redukcijo λ-izrazov s katerimi določimo vrstni red evaluacije redeksov λ-izraza. Vrstni red določimo lahko z vrstnim redom uporabe pravil kot tudi s pogoji, ki jih lahko postavimo v premisi pravil.

Osnovno pravilo evaluacije je β-redukcija s katerim evaluiramo funkcijo predstavljeno z λ-abstrakcijo na argumentu.

$$\frac{}{(\lambda x.M)\ N \rightarrow [N/x]M} \tag{6.4}$$

Zgornje pravilo nima nobenega pogoja v premisi. Uporabljamo ga lahko v primeru, da izraza M ali N nista evaluirana. Lahko zahtevamo, da sta izraza M ali N vrednosti tako, da dodamo v premiso pogoja M val in N val.

$$\frac{M \text{ val} \quad N \text{ val}}{(\lambda x.M)\ N \rightarrow [N/x]M} \tag{6.5}$$

Z uporabo pogojev v premisi lahko izbiramo strategijo evaluacije λ izraza. Strategijo evaluacije lahko definiramo tudi z naslednjima usmerjevalnima praviloma.

$$\frac{M \rightarrow M'}{M\ N \rightarrow M'\ N} \qquad \frac{N \rightarrow N'}{M\ N \rightarrow M\ N'} \tag{6.6}$$

Strategijo lahko določimo z vrstnim redom uporabe zgornjih pravil. Če uporabljamo najprej prvo pravilo potem s tem zagotovimo, da se telo λ-izraz, ki predstavlja funkcijo ovrednoti do vrednosti t.j. λ-abstrakcija, ki ne vsebuje redeksov.

Če uporabljamo najprej drugo pravilo potem najprej ovrednotimo argumente funkcijskega klica. Na ta način se λ-abstrakcija aplicira nad argumentom, ki predstavlja vrednost.

Evaluacijske strategije

Recimo, da λ-izraz vsebuje več redeksov. Imamo več različnih strategij, ki določajo vrstni red evaluacije redeksov v λ izrazu. Poglejmo si primer.

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

Izraz vsebuje tri redekse. Prepišimo izraz v bolj berljivo obliko in si oglejmo redekse.

$$\begin{array}{l} \underline{id\ (id\ (\lambda z.\ id\ z))} \\ id\ (\underline{id\ (\lambda z.\ id\ z)}) \\ id\ (id\ (\lambda z.\ \underline{id\ z})) \end{array}$$

Strategija določa kateri redeks se bo reduciral naslednji. Pogledali si bomo strategije: polna β-redukcija, normalna oblika, klic po imenu in klic po vrednosti.

1) *Polna β-redukcija* je strategija, kjer lahko reduciramo poljuben redeks v vsaki točki izbire. Če izberemo najprej notranje redekse potem dobimo naslednjo evaluacijo.

$$\begin{array}{ll} & id\ (id\ (\lambda z.\ \underline{id\ z})) \\ \rightarrow & id\ (\underline{id\ (\lambda z.z)}) \\ \rightarrow & \underline{id\ (\lambda z.z)} \\ \rightarrow & \lambda z.z \end{array}$$

2) *Strategija normalne oblike* izbere za redukcijo vedno skrajno levi, zunanji redeks. Po tej strategiji bi izraz iz prejšnjega primera reducirali takole.

$$\begin{array}{ll} & \underline{id\ (id\ (\lambda z.\ id\ z))} \\ \rightarrow & \underline{id\ (\lambda z.\ id\ z)} \\ \rightarrow & \lambda z.\ \underline{id\ z} \\ \rightarrow & \lambda z.z \end{array}$$

Z uporabo te strategije je evaluacijska relacija (parcialna) funkcija. Vsak izraz M se evaluira v natančno en izraz M'.

To strategijo kot tudi naslednjo imenujemo tudi *lena* strategija, ker evaluira argumente samo takrat, ko je to res potrebno.

3) *Strategija klic po imenu* ne dovoli redukcij znotraj abstrakcij. Sicer je enaka strategiji normalne oblike. Po tej strategiji bi dobili naslednjo redukcijo.

$$\begin{array}{ll} & \underline{id\ (id\ (\lambda z.\ id\ z))} \\ \rightarrow & \underline{id\ (\lambda z.\ id\ z)} \\ \rightarrow & \lambda z.\ \underline{id\ z} \end{array}$$

Evaluacija se je ustavila, ker je tretji redeks znotraj λ-abstrakcije. Zadnji izraz je torej normalna oblika po tej strategiji.

Haskell uporablja varianto te strategije, ki jo imenujejo klic po potrebi. Namesto, da bi se argument funkcije vedno znova evaluiral se izračuna ob prvem klicu. Vrednost rezultata evaluacije zamenja vse pojavitve izraza.

Ta strategija zahteva delo s sintaksnimi grafi—več izrazov ima lahko skupen podizraz.

4) Najbolj razširjena strategija je *klic po vrednosti*. Po tej strategiji se vedno reducira zunanji redeks, vendar po tem, ko so evaluirani vsi redeksi na desni.

$$\begin{array}{ll} & \underline{id\ (id\ (\lambda z.\ id\ z))} \\ \rightarrow & \underline{id\ (\lambda z.\ id\ z)} \\ \rightarrow & \lambda z.\ id\ z \end{array}$$

Opazimo, da v drugi vrstici argument $(\lambda z.\ id\ z)$ ni evaluiran pred celotnim izrazom, ker ni redeks—argument je sama λ-abstrakcija, ki vsebuje redeks $id\ z$.

Strategijo klic-po-vrednosti imenujemo *striktno*, ker so argumenti funkcije vedno evaluirani do vrednosti pred evaluacijo klica funkcije. Argumenti se torej ovrednotijo v primeru, da so potrebni pri izvajanju ali ne. *Ne-striktne* strategije kot je npr. klic-po-imenu evaluirajo samo argumente, ki so dejansko potrebni.

Klic po vrednosti

Strategija klic-po-vrednosti je največkrat privzeta metoda za evaluacijo λ-izazov. Strategija je definirana z naslednjimi pravili.

$$\frac{M \rightarrow M'}{M\ N \rightarrow M'\ N} \tag{6.7}$$

$$\frac{M \text{ val} \quad N \rightarrow N'}{M\ N \rightarrow M\ N'} \tag{6.8}$$

$$\frac{N \text{ val}}{(\lambda x.M)\ N \rightarrow [N/x]M} \tag{6.9}$$

Pravilo 6.7, ki je na prvem mestu, zahteva, da ovrednotimo najprej levo stran aplikacije. V čistem lambda računu so lambda abstrakcije edina oblika vrednosti. V primeru, da se M reducira v vrednost, potem to mora biti lambda abstrakcija.

Pravilo 6.8 poskrbi, da se po ovrednotenju leve strani ovrednoti še desna stran aplikacije. Šele ko tudi desna stran postane vrednost lahko apliciramo pravilo 6.9.

Ker N v pravilu 6.9 mora biti vrednost se lahko pravilo ujame samo v primeru, da je desna stran izraza vrednost.

Vrstni red proženja pravil popolnoma določa vrstni red ovrednotenja aplikacije $M\ N$:

1. M reduciramo v vrednost,
2. N reduciramo v vrednost in
3. izvedemo aplikacijo.

Klic po imenu

Strategija evaluacije klic-po-imenu je definirana s sledečimi pravili.

$$\frac{M \to M'}{M\ N \to M'\ N} \tag{6.10}$$

$$\frac{}{(\lambda x.M)\ N \to [N/x]M} \tag{6.11}$$

Aplikacija je zdaj predstavljena z enim samim pravilom 6.10. Pravilo ovrednoti levo stran aplikacije in šele nato začnemo z evaluacijo lambda abstrakcij. Za razliko od pravila 6.9, pravilo 6.11 nima pogoj, da je N vrednost.

Vaja 6.5.1. *Spreminjaj vrstni red pravil in opazuj kako se spreminja evaluacijska strategija pravil.* □

6.5.2 Church-Rosserjeva lastnost λ-računa

Church-Rosserjeva lastnost Λ pravi, da če sta dva λ-izraza β-konvertibilna potem imamo λ-izraz v katerega se oba izraza reducirata.

Definicija 6.5.1. *1. Binarna relacija R na Λ je* kompatibilna *(z operacijami) če*

$$\begin{array}{rcl} M\ R\ N & \Rightarrow & (ZM)\ R\ (ZN) \\ & & (MZ)\ R\ (NZ) \\ & & (\lambda x.M)\ R\ (\lambda x.N) \end{array}$$

2. Relacija kongruence *na Λ je kompatibilna ekvivalenčna relacija.*

3. Relacija redukcije *na Λ je kompatibilna, refleksivna in tranzitivna.*

Definicija 6.5.2. **(i)** *En korak β-redukcije $\to_\beta$ je definiran z naslednimi pravili.*

1. $(\lambda x.M)N \to_\beta [N/x]M$

2. $M \to_\beta N \Rightarrow ZM \to_\beta ZN, MZ \to_\beta NZ$ *in* $\lambda x.M \to_\beta \lambda x.N$

(ii) *β-redukcija $\to_\beta^*$ je definirana z naslednimi pravili.*

1. $M \to_\beta^* M$
2. $M \to_\beta N \Rightarrow M \to_\beta^* N$
3. $M \to_\beta^* N \wedge N \to_\beta^* L \Rightarrow M \to_\beta^* L$

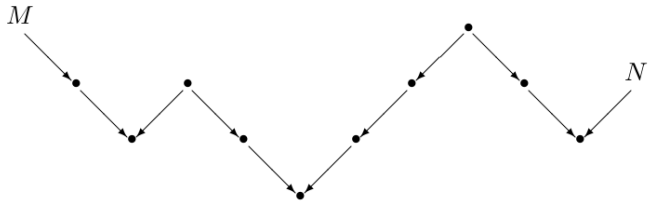
(iii) *Relacija β-konvertibilnost $= \beta$ je definirana z naslednimi pravili.*

1. $M \to_\beta^* N \Rightarrow M =_\beta N$
2. $M =_\beta N \Rightarrow N =_\beta M$
3. $M =_\beta N \wedge N =_\beta L \Rightarrow M =_\beta L$

Primer 6.5.1.

$$\omega \equiv \lambda x.x\ x$$
$$\Omega \equiv \omega\ \omega$$

Velja $KI\Omega \to_\beta^* I$. □



Slika 6.1: Pot med M in N

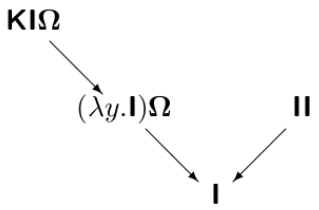
Intuitivno je $M =_\beta N$, če je M povezan z N preko povezav $\to_\beta$, kjer smer puščic ni pomemben. Povezava med M in N je prikazana na Sliki 6.1.

Primer 6.5.2. *Pokažimo, da velja $KI\Omega =_\beta II$. Izpeljava je prikazana na Sliki 6.2.* □

Definicija 6.5.3.

1. *β-redeks je izraz oblike $(\lambda x.M)N$. Izraz $[N/x]M$ je kontraktum redeksa.*
2. *λ-izraz je v β-normalni obliki, če nobeden pod-izraz ne vsebuje β-redeksa.*
3. *λ-izraz M* ima *β-normalno obliko, če $\exists N : M =_\beta N$ in je N v β-normalni obliki.*

Primer 6.5.3. *Izraz $(\lambda xx)y$ ni v β-no vendar ima izraz v β-no yy.*



Slika 6.2: Izpeljava $KI\Omega =_\beta II$

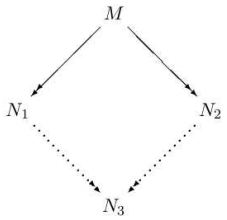
Lema 6.5.1. *Naj bo M v β-normalni obliki. Potem velja*

$$M \to_\beta^* N \Rightarrow N \equiv M$$

Dokaz. Očitno. □

Izrek 6.5.1 (Church-Rosserjev izrek)**.** *Če velja $M \to_\beta^* N_1$ in $M \to_\beta^* N_2$ potem $\exists N_3$: $N_1 \to_\beta^* N_3$ in $N_2 \to_\beta^* N_3$.*

Slika 6.3 prikazuje izpeljavo M grafično.

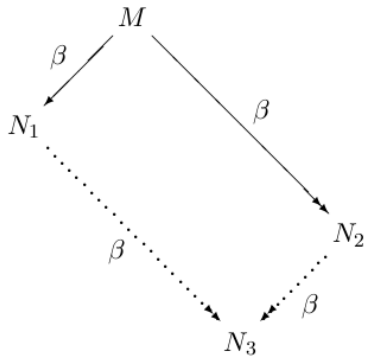


Slika 6.3: Grafična predstavitev Church-Rosserjevega izreka

V nadaljevanju bomo postopno dokazali Church-Rosserjev izrek. Osnovna ideja je naslednja. Da bi dokazali izrek je zadosti, da pokažemo da velja t.i. Strip Lema, ki je grafično prikazana na Sliki 6.4.

Naj bo $M \to_\beta N_1$ en korak redukcije, ki spremeni redeks R iz M v njegov kontraktum R' v N_1. Če sledimo spremembam redeksov R med redukcijo $M \to_\beta N_2$, potem najdemo N_3 z reduciranjem vseh preostalih redeksov R v N_2.

Da bi lahko sledili spremembam izrazov razširimo Λ z označenimi izrazi $\underline{\Lambda} \supseteq \Lambda$. Zaradi razširjene množice $\underline{\Lambda}$ je potrebno razširiti tudi pravila za $\underline{\beta}$-redukcijo.



Slika 6.4: Strip Lema

Definicija 6.5.4 (Označevanje)**.** *1. $\underline{\Lambda}$ je množica izrazov induktivno definirana na sledeč način.*

$$\begin{array}{rcl} x \in V & \Rightarrow & x \in \underline{\Lambda} \\ M, N \in \underline{\Lambda} & \Rightarrow & (M\ N) \in \underline{\Lambda} \\ M \in \underline{\Lambda}, x \in V & \Rightarrow & (\lambda x.M) \in \underline{\Lambda} \\ M, N \in \underline{\Lambda}, x \in V & \Rightarrow & ((\underline{\lambda} x.M)N) \in \underline{\Lambda} \end{array}$$

2. Označeni relaciji redukcije $\rightarrow_{\underline{\beta}}$ in $\rightarrow^*_{\underline{\beta}}$ sta razširjeni za označene izraze.

$$\begin{array}{rcl} (\lambda x.M)\ N & \rightarrow_{\underline{\beta}} & [N/x]M \\ (\underline{\lambda} x.M)\ N & \rightarrow_{\underline{\beta}} & [N/x]M \end{array}$$

Relacijo $\rightarrow_{\underline{\beta}}$ razširimo do kompatibilne operacije. Relacija $\rightarrow^*_{\underline{\beta}}$ je refleksivno in tranzitivno zaprtje $\rightarrow_{\underline{\beta}}$.

3. Če $M \in \underline{\Lambda}$ potem $|M| \in \Lambda$ dobimo tako, da odstranimo oznake. □

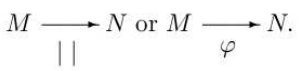
Definicija 6.5.5. *Preslikava $\varphi : \underline{\Lambda} \rightarrow \Lambda$ je induktivno definirana z naslednjimi pravili.*

$$\begin{array}{rcl} \varphi(x) & \equiv & x \\ \varphi(MN) & \equiv & \varphi(M)\varphi(N) \\ \varphi(\lambda x.M) & \equiv & \lambda x.\varphi(M) \\ \varphi((\underline{\lambda} x.M)N) & \equiv & \varphi(M)[\varphi(N)/x] \end{array}$$

Funkcija φ skrči vse redekse, ki so podčrtani odznotraj navzven.

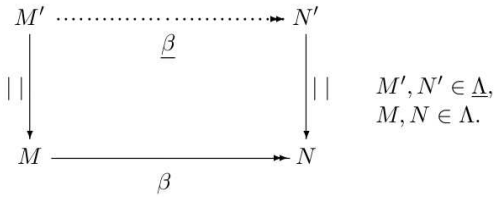
Opazka 6.5.1. *Zaradi strogega normalizacijskega izreka vrstni red krčenja redeksov ne vpliva na rezultat oz. katerakoli strategija redukcije λ-izraza vodi do istega rezultata. Normalizacijski izrek bo predstavljen kasneje.*

Notacija. Ekvivalenci $|M| \equiv N$ in $\varphi(M) \equiv N$ bomo grafično predstavili kot je prikazano na Sliki 6.5.



Slika 6.5: Notacija za $||$ in φ

Lema 6.5.2. *Slika 6.6.*



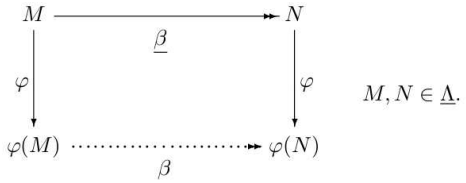
Slika 6.6: Lema $|| \circ \underline{\beta} \equiv \beta \circ ||$

Dokaz. Poglejmo primer, ko imamo samo en korak β-redukcije $M \to_\beta N$. N dobimo s krčenjem redeksa v M. N' lahko dobimo s krčenjem istega redeksa v M'. Splošen primer sledi po tranzitivnosti. □

Lema 6.5.3. **(i)** *Naj bo* $M, N \in \underline{\Lambda}$.

$$\varphi(M[N/x]) \equiv \varphi(M)[\varphi(N)/x]$$

(ii) *Slika 6.7.*



Slika 6.7: Lema $\varphi \circ \underline{\beta} \equiv \beta \circ \varphi$

Dokaz. **(i)** Indukcija po strukturi M. V vsakem od naslednjih primerov ekvivalenca velja, če po induktivni hipotezi predpostavimo pravilnost lastnosti za pod-izraze npr. M' in N'.

Primer 1. $M = y$.

$$\begin{array}{rcl} \varphi(y[N/x]) & \equiv & \varphi(y)[\varphi(N)/x] \\ x = y & \Rightarrow & \varphi(N) \equiv \varphi(N) \\ x \neq y & \Rightarrow & y = y \end{array}$$

Primer 2. $M = M'N'$.

$$\begin{array}{rcl} \varphi((M'N')[N/x]) & \equiv & \varphi(M'N')[\varphi(N)/x] \\ \varphi(M'[N/x]\ N'[N/x]) & \equiv & (\varphi(M')\ \varphi(N'))[\varphi(N)/x] \\ \varphi(M'[N/x])\ \varphi(N'[N/x]) & \equiv & \varphi(M')[\varphi(N)/x]\ \varphi(N')[\varphi(N)/x] \end{array}$$

Primer 3. $M = \lambda y.M'$

$$\begin{array}{rcl} \varphi((\lambda y.M')[N/x]) & \equiv & \varphi(\lambda y.M')[\varphi(N)/x] \\ \varphi(\lambda y.M'[N/x]) & \equiv & (\lambda y.\varphi(M'))[\varphi(N)/x] \\ \lambda y.\varphi(M'[N/x]) & \equiv & \lambda y.\varphi(M')[\varphi(N)/x] \end{array}$$

Primer 4. $M = (\underline{\lambda} y.M')N'$

$$\begin{array}{rcl} \varphi(((\underline{\lambda} y.M')N')[N/x]) & \equiv & \varphi(((\underline{\lambda} y.M')N')[\varphi(N)/x] \\ \varphi((\underline{\lambda} y.M')[N/x]\ N'[N/x]) & \equiv & (\varphi(\underline{\lambda} y.M')\ \varphi(N'))[\varphi(N)/x] \\ (\underline{\lambda} y.\varphi(M'[N/x]))\ \varphi(N'[N/x]) & \equiv & (\underline{\lambda} y.\varphi(M')[\varphi(N)/x])\ \varphi(N')[\varphi(N)/x] \end{array}$$

(ii) Ker $M, N \in \underline{\Lambda}$ potem predpostavljamo, da so nekateri redeksi v M, N označeni.

Uporabimo indukcijo po konstrukciji $\rightarrow_{\underline{\beta}}^{*}$ in (i).

Primer 1. $M \rightarrow_{\underline{\beta}}^{*} M$

Velja $N = M$ in tudi $\varphi(N) = \varphi(M)$.

Primer 2. $M \rightarrow_{\underline{\beta}} N \Rightarrow M \rightarrow_{\underline{\beta}}^{*} N$

Imamo en sam korak redukcije $M \rightarrow_{\underline{\beta}} N$.

Če je uporabljeno pravilo redukcije $(\underline{\lambda} x.M')N' \rightarrow_{\underline{\beta}} [N'/x]M'$ potem je lastnost dokazana z uporabo (i):

$$\begin{array}{rcll} \varphi((\underline{\lambda} x.M')N') & \equiv & \varphi(M'[N'/x]) & \\ (\underline{\lambda} x.\varphi(M'))\varphi(N') & \equiv & \varphi(M')[\varphi(N')/x] & \text{zaradi (i)} \\ \varphi(M')[\varphi(N')/x] & \equiv & \varphi(M')[\varphi(N')/x] & \end{array}$$

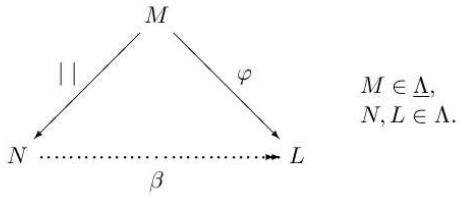
Za neoznačeno verzijo pravila velja isto.

Druga možnost je, da je bilo uporabljeno eno izmed pravil $M \rightarrow_{\underline{\beta}} N \Rightarrow ZM \rightarrow_{\underline{\beta}} ZN$, $MZ \rightarrow_{\underline{\beta}} NZ$ in $\lambda x.M \rightarrow_{\underline{\beta}} \lambda x.N$. Poglejmo $\underline{ZM} \rightarrow_{\underline{\beta}} ZN$. Po induktivni predpostavki velja lastnost za $M \rightarrow_{\underline{\beta}} N$: $\varphi(\underline{\beta}(M)) \equiv \beta(\varphi(M))$. Potem velja tudi: $\varphi(\underline{\beta}(ZM)) \equiv \beta(\varphi(ZM)$.

Primer 3. $M \to^*_{\underline{\beta}} N \wedge N \to^*_{\underline{\beta}} L \Rightarrow M \to^*_{\underline{\beta}} L$
Predpostavimo, da za obe redukciji velja lastnost prikazana na Sliki 6.7 in združimo obe poti.

□

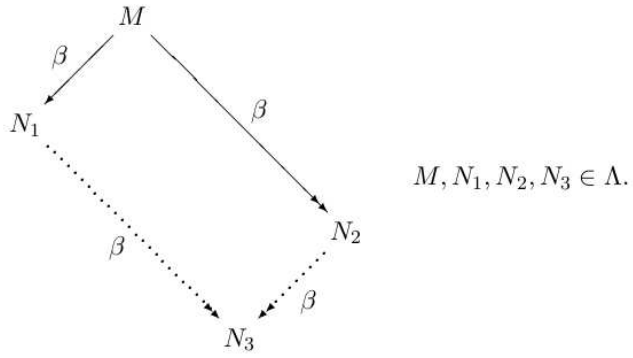
Lema 6.5.4. *Slika 6.8.*



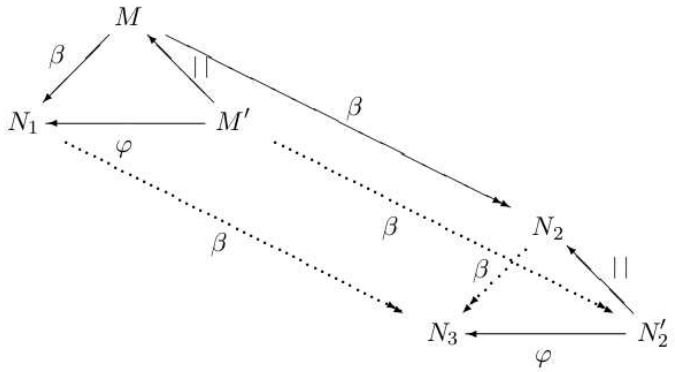
Slika 6.8: Lema $\beta(|M|) \equiv \varphi(M)$

Dokaz. Vsak redeks iz $|M|$, ki je reduciran med izvajanjem β je označen v M.
V dokazu uporabimo indukcijo glede na strukturo M. □

Lema 6.5.5 (Strip Lema)**.** *Slika 6.9.*



Slika 6.9: Strip Lema



Dokaz.

Slika 6.10: Dokaz Strip Leme

Naj bo N_1 rezultat redukcije primerka redeksa $R \equiv (\lambda x.P)Q$ v M. $M' \in \underline{\Lambda}$ dobimo iz M z zamenjavo R z $R' \equiv (\underline{\lambda} x.P)Q$. Potem velja $|M'| \equiv M$ in $\varphi(M') \equiv N_1$. Na osnovi lem 6.5.2, 6.5.3 in 6.5.4 lahko konstruiramo diagram na Sliki 6.10, ki dokazuje Strip Lemo. □

Opazka 6.5.2. *Osnovno idejo dokaza lahko opišemo na naslednji način. Za dane M, β-redukcijo $M \to_\beta N_1$ in sekvenco β-redukcij $M \to_\beta^* N_2$ pokažemo, da lahko konstruiramo $N_2 \to_\beta^* N_3$ in $N_1 \to_\beta^* N_3$. Te dve izpeljavi konstruiramo tako, da označimo v M' redeks, ki smo ga reducirali v M, da bi dobili N_1. Dano izpeljavo $M \to_\beta^* N_2$ izvedemo na M' in v primeru, da je v M' ostal označen redeks reduciramo še tega v izpeljavi $N_2 \to_\beta^* N_3$.*

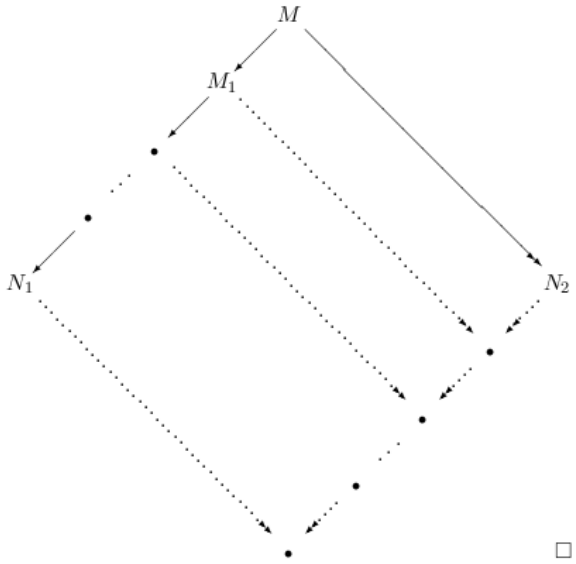
Naredili smo torej eno možno konstrukcijo izpeljave $M \to_\beta N_1 \to_\beta^* N_3$. Prej dokazani izreki nam omogočajo, da takšno konstrukcijo lahko naredimo.

Dokaz je potrebno gledati konstrukcijsko. Na primer, za dano sekvenco β-redukcij $M \to_\beta^* N_2$ obstaja *takšna sekvenca β-redukcij na povezavi $N_1 \to_\beta^* N_3$. V drugi izpeljavi lahko preprosto ponovimo redukcije vseh redeksov, ki so bili reducirani v prvi izpeljavi.* □

Poglejmo si zdaj še dokaz Church-Rosserjevega izreka.

Dokaz. Če $M \to_\beta^* N_1$ potem $M \equiv M_1 \to_\beta M_2 \to_\beta \ldots \to_\beta M_n \equiv N_1$. Lastnost CR velja na osnovi Strip Leme in diagrama na Sliki 6.11.

□



Slika 6.11: Dokaz CR

6.5.3 Posledice Church-Rosserjeve lastnosti

Church-Rosserjeva (okr. CR) lastnost jezika zagotavlja unikatnost normalne oblike neglede na obstoj normalne oblike.

Church-Rosserjeva lastnost ima pomen za λ-račun brez tipov, kjer normalizacijski izrek ne velja.

Posledica 6.5.1. *Če velja $M =_\beta N$ potem obstaja L tako da $M \to_\beta^* L$ in $N \to_\beta^* L$.*

Intuitivno dokažemo posledico takole: če imamo pot med M in N dano z $M =_\beta N$, potem ponavljaj uporabo CR na parih stavkov dokler ne dobiš L. Grafično je dokaz predstavljen na Sliki 6.12.

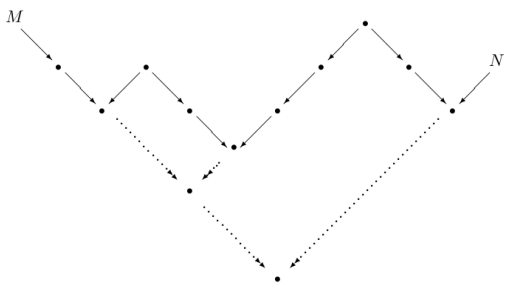
Dokaz. Indukcija na osnovi konstrukcije $=_\beta$.

Primer1. $M =_\beta N$ zaradi $M \to_\beta^* N$. Vzamemo $L \equiv N$.

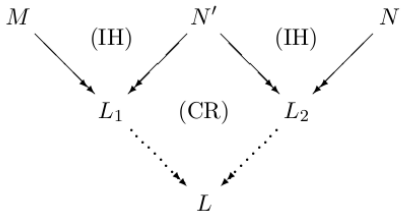
Primer 2 $M =_\beta N$ zaradi $N =_\beta M$. Po indukcijski hipotezi obstaja skupen β-redukt $M, N \to_\beta^* L_1$. Vzamemo $L \equiv L_1$.

Primer 3. $M =_\beta N$ zaradi $M =_\beta N'$, $N' =_\beta N$. Stanje je predstavljeno na Sliki 6.13.

□



Slika 6.12: Dokaz posledice CR za $M =_\beta N$



Slika 6.13: Dokaz posledice CR za $M =_\beta N$

Takojšnja posledica CR izreka je unikatnost normalne oblike za jezike, kjer velja CR.

Posledica 6.5.2. *Izraz t ima največ eno normalno obliko.*

Dokaz. Če velja $t \rightsquigarrow u, v$, kjer sta u in v v normalni obliki potem velja $u, v \rightsquigarrow w$ za nek w. Toda ker sta u in v v normalni obliki potem jih ni mogoče več naprej reducirati, velja mora torej $u = w = v$. □

Druga posledica CR je *konsistentnost* λ-računa: $\Lambda \not\vdash true =_\beta false$ ali, v splošnem, $\Lambda \not\vdash u =_\beta v$.

Konsistentnost jezika izhaja iz *logičnih* jezikov, kjer konsistentnost pomeni, da ni mogoče dokazati izjave in tudi negacije izjave.

Posledica 6.5.3. *λ-račun je konsistenten: ni res, da bi lahko dokazali enačbo $u =_\beta v$ za poljubna u in v.*

Dokaz.

- Če velja $u \rightsquigarrow v$ potem je enakost $u =_\beta v$ izpeljiva iz pravil za evaluacijo (redukcijo) in aksiomov enakosti.

- Obratno, če iz pravil za redukcijo in aksiomov enakosti sledi $u =_\beta v$ potem je enostavno videti, da obstajajo izrazi $U = t_0, t_1, \ldots, t_{2n-1}, t_{2n}$, tako da za $i \in [0..n-1]$ velja $t_{2i}, t_{2i+2} \rightsquigarrow t_{2i+1}$. S ponavljanjem aplikacije CR izreka končno dobimo $u, v \rightsquigarrow w$.

Če sta u in v dve različni normalni obliki istega tipa, potem ne obstaja takšen w, da bi lahko dokazali $u =_\beta v$. CR izrek torej pokaže denotacijsko konsistenco sistema. □

6.6 Izrazna moč λ-računa

6.6.1 Osnovni pojmi teorije izračunljivosti

Izračunljive funkcije so osnovni objekt študija na področju *teorije izračunljivosti*, ki je novejše ime za *Teorijo rekurzivnosti* (angl. Recursion theory).

Izračunljive funkcije ustrezajo intuitivnem konceptu *algoritma*. Na osnovi izračunljivih funkcij lahko govorimo o izračunljivosti brez referenciranja na konkretni model računanja kot je npr. Turingov stroj ali registerski stroj.

Za izračunljive funkcije velikokrat uporabljamo izraz *efektivno izračunljive* funkcije— funkcije za katere obstaja konkretna metoda po kateri jih lahko izračunamo.

Church-Turingove teze pravijo, da so izračunljive funkcije natančno tiste funkcije, ki jih lahko izračunamo na Turingovem stroju t.j. mehanskem stroju za računanje, ki ima na razpolago neskončno prostora in časa.

Ločimo med *totalnimi izračunljivimi funkcijami* in *parcialnimi izračunljivimi funkcijami*. Totalne funkcije so tiste, ki so definirane za vse možne vhode oz. nabore parametrov. Totalne izračunljive funkcije ustrezajo tistim funkcijam za katere obstaja Turingov stroj, ki je definiran za vse vhode in se vedno ustavi z rezultatom.

Parcialne funkcije so tiste, ki niso definirane za vse vhode. Parcialne funkcije ustrezajo natančno tistim funkcijam za katere obstaja Turingov stroj, ki jih zna izračunati vendar ni nujno, da se stroj ustavi za vsak vhod. V nekaterih primerih se Turingov stroj bodisi ustavi v nedefiniram položaju ali se vrti v zanki.

Okoli terminologije

Pred odkritjem Turingovega stroja in preden je bil definiran λ-račun je prevladovalo mnenje, da so vse funkcije izračunljive v smislu *totalnih funkcij*. Iz tega razloga je bila *izračunljivost* najprej obravnavana kot *odločjiva* lastnost.

Po odkritju prvih jezikov, ki niso odločljivi (za katere se Turingov stroj ne ustavi) je terminologija deloma ostala takšna kot je bila. Na primer, *rekurziven jezik* je še zdaj mišljen *totalno* rekurziven jezik [15].

V letih okoli 1930 je pojem *rekurzivna* funkcija pomenil isto kot *izračunljiva* funkcija. Pojem rekurzivnosti je bil marsikdaj uporabljen kot sinonim za izračunljivost.

Splošne rekurzivne funkcije kot jih je definiral Gödel so totalno izračunljive funkcije oz. so ekvivalentne totalnemu Turingovem stroju [9].

Parcialne rekurzivne funkcije ustrezajo parcialnim izračunljivim funkcijam oz. jeziku Turingovega stroja, ki nujno da se ustavi za vsak vhod.

Moderne definicije

V nadaljevanju si bomo pogledali še moderne definicije osnovnih dveh razredov funkcij. Totalno izračunljive funkcije gledano iz strani množic ustrezajo *rekurzivnim jezikom*. Parcialno izračunljive funkcije pa ustrezajo *rekurzivno preštevnim jezikom*.

Rekurzivni jezik L je takšen jezik za katerega obstaja Turingov stroj M, ki sprejme besede $w \in L$ in zavrne besede $w \notin L$. Turingov stroj, ki implementira rekurziven jezik, se torej ustavi za vsako vhodno besedo.

Rekurzivne jezike imenujemo tudi *totalni Turingov* jezik ali *odločljiv* jezik.

Rekurzivno preštevni jezik L je takšen jezik za katerega obstaja Turingov stroj M, ki našteje vse besede $w \in L$.

Za rekurzivno prešteven jezik L obstaja Turingov stroj M, ki sprejme vse besede $w \in L$ vendar ni nujno, da se ustavi v primeru da $w \notin L$.

Rekurzivno preštevni jezik imenujemo tudi *izračunljivo prešteven jezik* ali *delno odločljiv jezik*.

6.6.2 Izračunljive funkcije in λ račun

Predstavili smo že funkcijo seštevanja in množenja na $\mathbb{N}$, ki je definirana v λ-računu z uporabo Churchovih števil. Tukaj bomo pokazali, da lahko v λ-računu izrazimo vse izračunljive funkcije.

Najprej bomo predstavili totalno izračunljive funkcije, ki ustrezajo totalnim Turingovim strojem oz. odločljivim jezikom. Potem bomo predstavili še parcialne izračunljive funkcije, ki ustrezajo Turingovem stroju.

Opazka 6.6.1. *Z rekurzivnimi funkcijami se je ukvarjal Gödel.*

- *Vse probleme lahko prevedemo na procesiranje celih števil podobno kot lahko vse probleme prevedemo na Turingov stroj.*
- *Kako definirati razred funkcij, ki so izračunljive?*
- *Gödel se je začetno ukvarjal s primitivno rekurzivnimi funkcijami, ki predstavlja zelo obsežen razred funkcij.*

- *Primitivna rekurzija predstavlja totalno funkcijo.*
- *Omejeni operator minimizacije je ekvivalenten primitivnim rekurzivnim funkcijam.*
- *Neomejeni operator minimizacije ali μ-operator omogoča izražanje* parcialnih *funkcij*.
- *Definicija razreda (totalno) rekurzivnih funkcij*.

□

V nadaljevanju bomo najprej predstavili λ-definitabilne funkcije—funkcije, ki jih lahko definiramo z λ-računom. Potem bomo definirali razred *totalnih izračunljivih funkcij* (Gödelove splošno rekurzivne funkcije) na osnovi začetnih funkcij in operatorjev kompozicije, primitivne rekurzije in minimizacije.

Sledil bo dokaz, da so totalno izračunljive funkcije λ-definibilne. Če operator μ-minizacije definiramo kot neomejen operator dobimo *parcialne izračunljive funkcije*. Pokazali bomo, so tudi te λ-definibilne.

Totalne izračunljive funkcije in λ-definibilnost

Naravna števila $\mathbb{N}$ lahko predstavimo s Churchovimi števili, ki smo jih opisali v prejšnji sekciji. Ogledali si bomo še malce drugačen način predstavitve $\mathbb{N}$ in Boolovih vrednosti $\mathbb{B}$.

Definicija 6.6.1. **(i)** *true* $\equiv K$, *false* $\equiv K^*$.

(ii) *Naj bo $B \in \mathbb{B}$. if stavek lahko potem predstavimo z*

$$BPQ.$$

Definicija 6.6.2. *Naj bo $M, N \in \Lambda$.*

$$[MN] \equiv \lambda z.zMN$$

Potem velja

$$[M, N]\textit{true} = M,$$

$$[M, N]\textit{false} = N,$$

in lahko uporabimo $[M, N]$ kot urejen par.

Pare $[M, N]$ lahko uporabimo za alternativno predstavitev celih števil.

Definicija 6.6.3. *Za vsak $n \in \mathbb{N}$ je število $\ulcorner n \urcorner$ definirano induktivno na sledeč način.*

$$\begin{array}{lcl} \ulcorner 0 \urcorner & \equiv & I \\ \ulcorner n+1 \urcorner & \equiv & [false, \ulcorner n \urcorner] \end{array}$$

Lema 6.6.1. *Obstajajo kombinatorji S^+, P^- in $Zero$ tako, da velja:*

$$\begin{array}{lcl} S^+ \ulcorner n \urcorner & \equiv & \ulcorner n+1 \urcorner \\ P^- \ulcorner n+1 \urcorner & \equiv & \ulcorner n \urcorner \\ Zero \ulcorner 0 \urcorner & \equiv & \textit{true} \\ Zero \ulcorner n+1 \urcorner & \equiv & \textit{false} \end{array}$$

Dokaz.

$$\begin{array}{lcl} S^+ & \equiv & \lambda x.[\text{false}, x], \\ P^- & \equiv & \lambda x.x\ \text{false}, \\ Zero & \equiv & \lambda x.x\ \text{true} \end{array}$$

□

Definicija 6.6.4. *(Lambda definibilnost)*

(i) *Numerična funkcija je preslikava*

$$\varphi : \mathbb{N}^p \to \mathbb{N}$$

za nek p. Število argumentov φ je p.

(ii) *Numerična funkcija φ z p argumenti je λ-definibilna, če za nek kombinator F velja*

$$F \ulcorner n_1 \urcorner \ldots \ulcorner n_p \urcorner = \ulcorner \varphi(n_1, \ldots, n_p) \urcorner$$

za vse $n, \ldots, n_p \in \mathbb{N}$. Če zgornja enačba velja potem je φ λ-definirana z F.

Definicija 6.6.5. Začetne funkcije *so numerične funkcije U_i^n, S^+, Z definirane kot*

$$\begin{array}{rcl} U_i^n(x_1, \ldots, x_n) & = & x_i, \quad (1 \leq i \leq n) \\ S^+(n) & = & n+1, \\ Z(n) & = & 0 \end{array}$$

Naj bo $P(n)$ numerična relacija. Izraz

$$\mu m[P(m)]$$

predstavlja najmanjše število m tako da velja $P(n)$, če obstaja takšno število; sicer je nedefinirana.

Definicija 6.6.6. *Naj bo $\mathcal{A}$ razred numeričnih funkcij.*

(i) *$\mathcal{A}$ je zaprt za kompozicijo, če velja za vse φ definirane kot*

$$\varphi(\vec{n}) = \chi(\psi_1(\vec{n}), \ldots, \psi_m(\vec{n})),$$

kjer so $\chi, \psi_1, \ldots, \psi_m \in \mathcal{A}$ potem tudi $\varphi \in \mathcal{A}$.

(ii) *$\mathcal{A}$ je zaprta za primitivno rekurzijo, če za vse φ definirane kot*

$$\begin{array}{rcl} \varphi(0, \vec{n}) & = & \chi(\vec{n}), \\ \varphi(k+1, \vec{n}) & = & \psi(\varphi(k, \vec{n}), k, \vec{n}), \end{array}$$

kjer $\chi, \psi \in \mathcal{A}$ velja $\varphi \in \mathcal{A}$.

(iii) *$\mathcal{A}$ je zaprt za minimizacijo, če za vse φ definirane kot*

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

kjer $\chi \in \mathcal{A}$ in

$$\forall \vec{n}\, \exists m\, \chi(\vec{n}, m) = 0,$$

velja $\varphi \in \mathcal{A}$.

Definicija 6.6.7. *Razred* totalno izračunljivih funkcij $\mathcal{R}$ *(ali rekurzivnih funkcij) je najmanjši razred numeričnih funkcij, ki vsebuje vse začetne funkcije in je zaprt za kompozicijo, primitivno rekurzijo in minimizacijo.*

Dokaz, da so vse izračunljive (totalne in parcialne) funkcije λ-definibilne je podal Kleene leta 1936.

Najprej bomo pokazali, da so vse totalno izračunljive funkcije (rekurzivne funkcije) λ-definitabilne.

Lema 6.6.2. *Začetne funkcije so λ-definibilne.*

Dokaz. Začetne funkcije definiramo takole:

$$\begin{array}{rcl} U_i^n & \equiv & \lambda x_1 \ldots x_n . x_i, \\ S^* & \equiv & \lambda x . [\mathsf{false}, x], \\ Z & \equiv & \lambda x . \ulcorner 0 \urcorner. \end{array}$$

□

Lema 6.6.3. *Λ-definibilne funkcije so zaprte za kompozicijo.*

Dokaz. Naj bodo $\chi, \psi_1, \ldots, \psi_m$ λ-definirane z $G, H_1, \ldots, H_m$. Potem

$$\varphi(\vec{n}) = \chi(\psi_1(\vec{n}), \ldots, \psi_m(\vec{n}))$$

λ-definiramo z

$$F = \lambda\vec{x}.G\,(H_1\,\vec{n})\ldots(H_m\,\vec{n}).$$

□

Primer 6.6.1. *Poglejmo si primer uporabe primitivne rekurzije. Funkcijo seštevanja definiramo takole.*

$$\begin{array}{rcl} Add(0,y) & = & y \\ Add(x+1,y) & = & 1 + Add(x,y) = S^+(Add(x,y)) \end{array}$$

Intuitivna definicija algoritma $Add(m,n)$ je sledeča. Če je $m = 0$ potem je rezultat n, sicer izračunaj $Add(m-1,n)$ in vrni naslednika od Add.

Potrebujemo torej izraz Add tako, da velja

$$Add\,x\,y = \textit{if}\ Zero\,0\ \textit{then}\ y\ \textit{else}\ S^+(Add(P^-x)y)$$

To enačbo lahko rešimo s kombinatorejem fiksna točka Y.

$$Add = Y(\lambda axy.\textit{if}\ Zero\,0\ \textit{then}\ y\ \textit{else}\ S^+(a(P^-x)y))$$

□

Lema 6.6.4. Λ*-definibilne funkcije so zaprte za primitivno rekurzijo.*

Dokaz. Na bo φ definirana na sledeč način.

$$\begin{array}{rcl} \varphi(0,\vec{n}) & = & \chi(\vec{n}) \\ \varphi(x+1,\vec{n}) & = & \psi(\varphi(k,\vec{n}),k,\vec{n}) \end{array}$$

Funkcije χ in ψ sta λ-definirani z G in H.

$$\begin{array}{rcl} F\,x\,\vec{y} & = & \text{if } Zero\,x \text{ then } G\,\vec{y} \text{ else } H(F(P^-x)\,\vec{y})\,(P^-x)\,\vec{y} \\ & \equiv & D(F,x,\vec{y}). \end{array}$$

Zadosti je poiskati takšen F za katerega velja

$$\begin{array}{rcl} F & = & \lambda x\vec{y}.D(F,x,\vec{y} \\ & = & (\lambda fx\vec{y}.D(f,x,\vec{y}))\ F \end{array}$$

Takšen F lahko najdemo zaradi izreka o fiksni točki. □

Lema 6.6.5. Λ*-definibilne funkcije so zaprte za minimalizacijo.*

Dokaz. Na bo φ definirana na sledeč način.

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

Funkcije χ je λ-definirana z G. Spet, zaradi izreka o fiksni točki obstaja izraz

$$\begin{array}{rcl} H\ \vec{x}\, y & = & \text{if } Zero(G\vec{x}y) \text{ then } y \text{ else } H\ \vec{x}\ (S^+ y) \\ & \equiv & (\lambda h \vec{x} y.E(h, \vec{x}, y))H\vec{x}y. \end{array}$$

Naj bo takšen $F \equiv \lambda\vec{x}.Hx\ulcorner 0\urcorner$. Potem F λ-definira φ:

$$\begin{array}{rcll} F\ulcorner\vec{n}\urcorner & = & H\ulcorner\vec{n}\urcorner\ulcorner 0\urcorner & \\ & = & \ulcorner 0\urcorner & \text{if } G\ulcorner\vec{n}\urcorner\ulcorner 0\urcorner = \ulcorner 0\urcorner \\ & = & H\ulcorner\vec{n}\urcorner\ulcorner 1\urcorner & \text{else} \\ & = & \ulcorner 1\urcorner & \text{if } G\ulcorner\vec{n}\urcorner\ulcorner 1\urcorner = \ulcorner 1\urcorner \\ & = & H\ulcorner\vec{n}\urcorner\ulcorner 2\urcorner & \text{else} \\ & = & \ulcorner 2\urcorner & \text{if } \ldots \\ & = & \ldots & \end{array}$$

□

Izrek 6.6.1. *Vse totalno izračunljive funkcije so λ-definibilne.*

Dokaz. Zaradi Lem 6.6.2-6.6.5. □

Lastnost velja tudi v obratno smer. Za numerične funkcije torej velja, da je φ totalno izračunljiva funkcija čče je φ λ-definibilna. Velja pa še več:

$$\varphi \text{ je parcialno izračunljiva} \Leftrightarrow \varphi \text{ je } \lambda\text{-definibilna.}$$

Dokaz zadnjih dveh lastnosti si lahko ogledate v [2].

6.7 Opombe

Klasična predstavitev λ-računa je podana v članku Barendregt in Barendsen, Introduction to Lambda Calculus [3]. Obsežnejša predstavitev, ki vključuje dokaze pomembnejših izrekov, se nahaja v knjigi Barendregt z naslovom The Lambda Calculus: Its Syntax and Semantics [2].