Predavanje 2

Lambda račun

Iztok Savnik, FAMNIT

Februar, 2023.

Literatura

Henk Barendregt, Erik Barendsen, Introduction to Lambda Calculus, March 2000.

Lambda račun

- Leibniz je imel naslednji ideal:
  1) Kreiraj ‘univerzalni jezik’ s katerim lahko izraziš vse možne probleme.
  2) Poišči postopek za reševanje problemov izraženih v univerzalnem jeziku.
- (1) je bilo razrešeno z:
  – Teorijo množic + predikatni račun (Frege,Russel,Zermelo)
- (2) je postal pomemben filozofski problem:
  – Ali lahko rešimo vse probleme izražene v univerzalnem jeziku?
  – Odločljivost (Entscheidungsproblem)

Entscheidungsproblem

- Negativen izid
- Alonzo Church, 1936
  - Predlaga λ-račun kot razširitev logike
  - Pokaže obstoj neodločljivega problema
  - Funkcijski programski jeziki
- Alan Turing, 1936
  - Predlaga svoj stroj (Turingov stroj)
  - Turing je dokazal, da oba modela definirata isti razred izračunljivih funkcij
  - Stroj ekvivalenten Von Neumann modelu računalnika
  - Imperativni programski jeziki

Funkcije

- Funkcija je osnovni koncept klasične in moderne matematike.
- Naj bodo A in B množice in naj bo f relacija.
  - $dom(f) = A$
  - $\forall x \in A$: $\exists$ natančno določen $y \in B$ tako da $(x,y) \in f$
  - Enoličnost: $(x,y) \in f \land (x,z) \in f \Rightarrow y=z$
  - *f preslika (transformira) x* v *y*
- $f : A \to B$
  - *f* je funkcija, ki slika *A* v *B*

Lambda notacija

- Lambda izrazi
  - Izrazi čistega lambda računa:
    - Spremenljivke: *x, y, z, ...*
    - Lambda abstrakcija: $\lambda x.M$
    - Aplikacija: $M\ N$
- Lambda abstrakcija $\lambda x.M$ *predstavlja* funkcijo
  - x je argument funkcije
  - M je funkcijski izraz
    - Recept, ki pove kako funkcijo »izračunam«
- Aplikacija $M\ N$
  - Če $M = \lambda x.M'$ potem so vse pojavitve x iz M' zamenjane z N
  - Mehanična definicija prenosa parametrov

Lambda notacija

- Naj bo $x + 1$ izraz s spremenljivko x
  - Matematična notacija: $f(x) = x + 1$
  - Lambda notacija: λx.(x + 1)
- Naj bo $x + y$ izraz kjer sta x in y spremenljivke
  - Matematična notacija: $f(x,y) = x + y$
  - Lambda notacija: $\lambda x.\lambda y.(x + y)$
- Očitna razlika:
  - *Λ*-notacija ne imenuje funkcij

Definicija sintakse λ-računa

Definicija: Množica λ-izrazov Λ je konstruirana iz neskončne množice spremenljivk *{v,v',v'',v''', ...}* z *aplikacijo* in *λ-abstrakcijo:*

$$x \in V \Rightarrow x \in \Lambda$$

$$M,N \in \Lambda \Rightarrow (M\ N) \in \Lambda$$

$$X \in V,\ M \in \Lambda \Rightarrow \lambda x.M \in \Lambda$$

Backus-Naur oblika sintakse λ-računa:

$$M ::= V \mid (\lambda v.M\ ) \mid (M\ N)$$

$$V ::= v,\ v'\ ,v''\ .\ .\ .$$

Syntax rules

- *Aplikacija* je levo asociativna

$$M\ N\ L \equiv (M\ N)\ L$$

- *Λ-abstrakcija* je desno asociativna

$$\lambda x.\lambda y.\lambda z.M\ N\ L \equiv \lambda x.(\lambda y.(\lambda z.((M\ N)\ L)))$$

- Pogosto uporabljamo naslednjo okrajšavo

$$\lambda xyz.M \equiv \lambda x.\lambda y.\lambda z.M$$

Primeri

- Poglejmo si nekaj primerov λ-izrazov (glej presledke)

$$y$$

$$y\ x$$

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

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

$$(\lambda u.\lambda f.\lambda x.f\ (u\ f\ x))\ (\lambda v.\lambda y.v\ y)$$

Primeri: λ in ocaml

Nekateri λ-izrazi (glej presledke!):

```
3
λx.x
(λx.x) (λy.y ∗ y)
(λz.z + 1) 3
```

OCaml:

```
# 3;;
- : int = 3
# function x -> x;;
- : ’a -> ’a = <fun>
# (function x -> x) (function y->y*y);;
- : int -> int = <fun>
# (function z -> z + 1) 3;;
- : int = 4
```

Proste in povezane spremenljivke

- Abstrakcija $\lambda x.M$ povezuje spremenljivko x v izrazu M
  - Na podoben način so argumenti funkcije povezani s telesom funkcije
- M je definicijsko območje spremenljivke x v izrazu $\lambda x.M$
- Spremenljivka x je *prosta* v izrazu M, če ne obstaja λ-abstrakcija, ki jo povezuje
- Ime proste spremenljivke je pomembno medtem, ko ime povezane spremenljivke ni pomembno.
- Primer:

$$\lambda x.(x + y)$$

Računanje prostih spremenljivk

Definicija: Množica prostih spremenljivk λ-izraza *M*, ki jo imenujemo *FV(M)*, je definirana z naslednjimi pravili:

$$FV(x) = \{x\}$$

$$FV(M\ N\ ) = FV(M) \cup FV(N)$$

$$FV(\lambda x.M) = FV(M) - \{x\}$$

Primer:

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

Definicija: λ-izraz *M* je zaprt, če $FV(M)=\{\}$.

Substitucija

- Substitucija predstavlja osnovo evaluacije λ-računa
  - Računanje je prepisovanje nizov znakov?
- Zamenjaj vse primerke sprem. x v λ-izrazu M z N:

$$[N/x]M$$

Definicija: Naj bosta M,N∈Λ in x,z∈V. Pravila substitucije so naslednja.

$$[N/x]x = N$$

$$[N/x]z = z, \text{ če } z \neq x$$

$$[N/x](L\ M) = ([N/x]L)([N/x]M)$$

$$[N/x](\lambda z.M) = \lambda z.([N/x]M), \text{ če } z \neq x \wedge z \notin FV(N)$$

Primer substitucije

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

$$\equiv \lambda z.(\lambda u.u)\ z\ (y\ (\lambda v.v))$$

- Preveri evaluacijo pravil substitucije!

α-konverzija

- Preimenovanje povezanih spremenljivk v λ-izrazu vodi do ekvivalentnega λ-izraza.
- Primer:

$$\lambda x.x \equiv \lambda y.y$$

- Pravilo α-konverzije:

$$\lambda x.M \ \equiv \ \lambda y.([y/x]M\ ),\ if\ y \notin FV(M).$$

Primer: α-konverzija

- Λ-izraz:

  (λf.λx.f (f x)) (λy.y + x)

- Analiza izraza
  - (λf.λx.f (f x)) -- x in f sta vezani spremenljivki
  - (λy.y + x) – y je vezana in x je prosta spremenljivka
  - Imamo dve pojavitvi spremenljivke x
    - Prostih spremenljivk ne moremo preimenovati.
  - Preimenujemo lahko x v (λf.λx.f (f x)).
- Preimenovanje
  - (λf.λx.f (f x)) ≡ λf.λz.[z/x]f (f x) ≡ λf.λz.f (f z)
  - Rezultat: (λf.λz.f (f z)) (λy.y + x)

Evaluacija

- Λ-račun je zelo izrazen jezik, ki je ekvivalenten Turingovem stroju.
- Evaluacija λ-izrazov je osnovana na :
  1) α-konverziji in

  2) substituciji
- Evaluacijo pogosto imenujemo redukcija
- Λ-izrazi so reducirani do vrednosti
  - Vrednosti so normalne oblike λ-izrazov: λ.izrazi, ki jih ne moremo naprej reducirati

β-redukcija

- β-redukcija je edino pravilo uporabljeno za evaluacijo čistega λ-računa (razen preimenovanja)
- Izraz $(\lambda x.M)\ N$ sestavlja operator $(\lambda x.M)$, ki je apliciran na parameteru N
- Intuitivna interpretacija $(\lambda x.M)\ N$ je substitucija x v M z N

β-redukcija

Definicija: Naj bo *λx.M* λ-izraz. Aplikacija *(λx.M)* na parametru *N* je izvedena z β-redukcijo:

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

- Izraz (λx.M) N imenujemo redeks (angl. redex = reducable expression)
- Izraz [N/x]M imenujemo kontraktum.

β-redukcija

P vsebuje redeks *(λx.M) N, ki se ovrednoti v [N/x]M in tako dobimo P'*

Pravimo, da *P* *β-reducira* v *P':*

$$P \rightarrow_\beta P'$$

***Definicija**: β-izpeljava je sestavljena iz ene ali večih β-redukcij.* *Β-izpeljava* iz *M v N:*

$$M \twoheadrightarrow_\beta N$$

β-normalna oblika

Definicija:

1) λ-izraz *Q,* ki ne vsebuje *β-redeksov je v β-normalni obliki.*
2) Razred vseh *β-normalnih oblik* imenujemo *β-nf.*
3) Če *P* β-reducira v *Q,* ki je β-nf, potem je *Q* β-normalna oblika *P.*

Primeri evaluacije

- $(\lambda x.x\ y)\ (u\ v) \to_\beta u\ v\ y$
- $(\lambda x.\lambda y.x)\ z\ w \to_\beta (\lambda y.z)\ w \to_\beta z$

  $(\lambda x.\lambda y.x)\ z\ w \twoheadrightarrow_\beta z$
- $(\lambda x.(\lambda y.yx)\ z)\ v \to [v/x](\lambda y.yx)\ z = (\lambda y.yv)\ z$

  $\to [z/y]yv = zv$

  $(\lambda x.(\lambda y.yx)\ z)\ v \twoheadrightarrow_\beta zv$

Primer: α-konverzija in β-redukcija

- Λ-izraz:

$$(\lambda f.\lambda x.f\ (f\ x))\ (\lambda y.y + x)$$

- Slepa substitucija vodi do:

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

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

$$= \lambda x.x + x + x$$

- Korektna substitucija:

$$(\lambda f.\lambda z.f\ (f\ z))\ (\lambda y.y + x)$$

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

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

$$= \lambda z.z + x + x$$

Primeri evaluacije

- Primer s funkcijo identitete

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

- Še en primer s funkcijo identitete

  $(\lambda f.f\ (\lambda x.x))(\lambda x.x) \rightarrow$

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

  $(\lambda x.x)(\lambda y.y) \rightarrow$

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

Primeri evaluacije

- Ponavljajoča se *β-izpeljava:*

  (λx.xx)(λy.yy)

  → [(λy.yy)/x]xx = (λx.xx)(λy.yy)

  → [(λy.yy)/x]xx = (λx.xx)(λy.yy)

  → ...

- *β-izpeljava,* ki šteje*:*

  (λx.xxy)(λx.xxy)

  → [(λx.xxy)/x]xxy = (λx.xxy)(λx.xxy)y

  → ([(λx.xxy)/x]xxy)y = (λx.xxy)(λx.xxy)yy → …

Funkcije višjega reda

- Funkcija višjega reda je funkcija:
  - Uporabi drugo funkcijo kot argument in/ali vrne funkcijo kot rezultat aplikacije funkcije.
- Primer:
  - Konstruiraj kompozitum: $(f \circ f)(x) = f(f(x))$
  - Lambda izraz: $\lambda f.\lambda x.f\ (f\ x)$

$$
\begin{aligned}
& (\lambda f.\lambda x.f\ (f\ x))(\lambda y.y + 1) \\
= {} & \lambda x.(\lambda y.y + 1)((\lambda y.y + 1)\ x) \\
= {} & \lambda x.(\lambda y.y + 1)(x + 1) \\
= {} & \lambda x.(x + 1) + 1
\end{aligned}
$$

Funkcije višjega reda

- Funkcija (f∘f)(x) v Lispu

```
    (lambda(f)(lambda(x)(f (f x))))

    ((lambda(f)(lambda(x)(f (f x))))(lambda(y)(+ y 1))
  = (lambda(x)((lambda(y)(+ y 1))((lambda(y)(+ y 1)) x))))
  = (lambda(x)((lambda(y)(+ y 1))(+ x 1))))
  = (lambda(x)(+ (+ x 1) 1))
```

- Funkcija konstruira novo funkcijo iz funkcije, ki je bila podana kot parameter.

Primeri v Ocaml

```
# let c = 4;;
val c : int = 4
# let sq = function x -> x*x;;      (* λx.x*x *)
val sq : int -> int = <fun>
# let nx = function x -> x + 1;;    (* λx.x+1 *)
val nx : int -> int = <fun>

# let compose1 = function f -> function x -> f(f(x));;                  (* λf.λx.f(f(x)) *)
val compose1 : ('a -> 'a) -> 'a -> 'a = <fun>
# let compose = function f -> function g -> function x -> f(g(x));;   (* λf.λg.λx.f(g(x)) *)
val compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
# let rcompose = function f -> function g -> function x -> g(f(x));;  (* λf.λg.λx.g(f(x)) *)
val rcompose : ('a -> 'b) -> ('b -> 'c) -> 'a -> 'c = <fun>

# (compose nx nx) 3;;
- : int = 5
# (compose sq nx) 3;;
- : int = 16
# (rcompose sq nx) 3;;
- : int = 10
```

Programiranje v λ-računu

- Funkcije v Curry obliki
- Kombinatorji
  - Primitivi programskih jezikov
- Logične vrednosti
  - If stavek
- Cela števila
  - Aritmetika
- Rekurzija

Curry funkcija

- V λ-računu ima funkcija en sam parameter
- Več parametrov lahko implementiramo s funkcijami višjega reda
- F je funkcija s parametroma (x, y) in telesom M
  - $F(x,y) = M$ | M je izraz s prostima sprem. x in y
  - Klic $F(N,L)$ | Želimo zamenjati x z N in y z L
- Curry notacija: $F \equiv \lambda x.\lambda y.M$
  - $F\ N\ L \rightarrow (\lambda y.[N/x]M\ )L \rightarrow [L/y][N/x]M$
  - Λ-račun s pari: $F \equiv \lambda\langle x,y\rangle.M$
- Prehod iz $\lambda\langle x,y\rangle.M$ v $\lambda x.\lambda y.M$ je pretvorba v Curry obliko (angl. Currying)

Primer: Curry funkcije

- Običajna notacija: sum ≡ $\lambda\langle x,y\rangle.x + y$
  - sum : $\mathbb{Z}\rightarrow\mathbb{Z}\rightarrow\mathbb{Z}$ (tip funkcije sum)
- Curry notacija: sum ≡ $\lambda x.\lambda y.x + y$
  - Aplikacija enega parametra vrne funkcijo.
  - suma ≡ $(\lambda x.\lambda y.x + y)\ a$ -> $\lambda y.a + y$
  - suma : $\mathbb{Z}\rightarrow\mathbb{Z}$
- Ocaml knjižnice so napisane v Curry obliki
  - Možno definirati nove funkcije na osnovi obstoječih.
  - Ogledali si bomo primere.

Kombinatorji

- Kombinatorji so primitivne funkcije
  - Predstavljajo osnovne operacije računanja
  - Funkcije: indentiteta, kompozicija, izbira, itd.
- Kombinatorska logika CL
  - Curry, Feys, 1958
  - Kombinatorji so gradniki CL
  - CL uporabi kombinatorje **I**, **K** in **S**
- Kombinatorji se uporabljajo za pisanje funkcij višjega reda
  - Kombinatorji konstruirajo nove funkcije
    - Pri predstavitvi funkcijskih jezikov bomo videli primere!
  - Funkcije višjega reda: apply, map, fold, filter, itd.

Kombinatorji

- Funkcija identitete:

  $\mathbf{I} = \lambda x.x$

- Izbira enega argumenta od dveh:

  $\mathbf{K} = \lambda x.(\lambda y.x)$

- Podajanje argumenta dvem funkcijam:

  $\mathbf{S} = \lambda x.\lambda y.\lambda z.(x\ z)(y\ z)$

- Funkcija, ki se ponavlja:

  $\mathbf{\Omega} = (\lambda x.x\ x)(\lambda x.x\ x)$

- Kompozicija funkcij:

  $\mathbf{B} = \lambda f.\lambda g.\lambda x.f(g\ x)$

Kombinatorji

- Inverzna kompozicija funkcij:

  $$\mathbf{B'} = \lambda f.\lambda g.\lambda x.g(f\ x)$$

- Podvojevanje argumenta funkcije:

  $$\mathbf{W} = \lambda f.\lambda x.f\ x\ x$$

- Rekurzivna funkcija:

  $$\mathbf{Y} = \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x))$$

Logične vrednosti

- Kako predstaviti logične vrednosti?
  - true ≡ $\lambda t.\lambda f.t$ | funkcija vrne prvi od dveh argumentov
  - false ≡ $\lambda t.\lambda f.f$ | funckija vrne drugi od dveh argumentov
- IF stavek je enostavna aplikacija logične vrednosti
  - $\lambda l.\lambda m.\lambda n.\ l\ m\ n$
  - Logična vrednost določi prvi ali drugi argument
- Evaluacija IF stavka

$$
\begin{aligned}
\text{IF true } M\ N \equiv\ & (\lambda l.\lambda m.\lambda n.\ l\ m\ n)\ \text{true}\ M\ N \rightarrow \\
& (\lambda m.\lambda n.\ \text{true}\ m\ n)\ M\ N \rightarrow \\
& \text{true}\ M\ N = (\lambda t.\lambda f.t)\ M\ N \rightarrow \\
& (\lambda f.M\ )\ N \rightarrow M
\end{aligned}
$$

Church-ova števila

$C_0 = \lambda z.\lambda s.z$

$C_1 = \lambda z.\lambda s.s\ z$

$C_2 = \lambda z.\lambda s.s(s\ z)$

...

$C_n = \lambda z.\lambda s.s(s(. . . (s\ z) . . .)$

- Peanovi aksiomi
  - $0 \in \mathbb{N}_0$
  - $n \in \mathbb{N}_0 \Rightarrow n+1 \in \mathbb{N}_0$
- Število n je predstavljeno kot C_n
  - $n = 0+1+...+1$ | n kratni naslednik 0
  - z predstavlja ničlo in s predstavlja funkcijo naslednik
- Aritmetične operacije
  - Plus = $\lambda m.\lambda n.\lambda z.\lambda s.m\ (n\ z\ s)\ s$
  - Times = $\lambda m.\lambda n.m\ C_0\ (\text{Plus}\ n)$

Church-ova števila

$(\text{Plus } 1\ 2) \to^* 3$

$\text{Plus } (\lambda z.\lambda s.s\ z)\ (\lambda z.\lambda s.s(s\ z)) \to$

$(\lambda m.\lambda n.\lambda z.\lambda s.m(n\ z\ s)s)\ (\lambda z.\lambda s.s\ z)\ (\lambda z.\lambda s.s(s\ z)) \to$

$(\lambda n.\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)(n\ z\ s)s)\ (\lambda z.\lambda s.s(s\ z)) \to$

$\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)((\lambda z.\lambda s.s(s\ z))\ z\ s)s \to$

$\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)((\lambda s.s(s\ z))\ s)s \to$

$\lambda z.\lambda s.(\lambda z.\lambda s.s\ z)(s(s\ z))s =$

$\lambda z.\lambda s.(((\lambda z.\lambda s.s\ z)\ (s(s\ z)))s) \to$

$\lambda z.\lambda s.((\lambda s.s(s\ (s\ z)))s) \to$

$\lambda z.\lambda s.s(s\ (s\ z))$

Rekurzija

- Rekurzijo lahko izrazimo s kombinatorjem **Y**
  - $Y = \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x))$
- Pomembna lastnost **Y**
  - $Y\ F =_\beta F\ (Y\ F)$
  - Dokaz:

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

Rekurzija

```
# let rec fact n = if n=0 then 1 else n * fact (n-1);;
val fact : int -> int = <fun>
```

- Operacija fakulteta: n!
  - Intuitivna definicija
- Definicija rekurzivne funkcije F
  - $G = \lambda f.M$ | M je telo f
  - $F = Y\ G$
- Izpeljava F

```
if n = 0 then 1
else n ∗ (if n − 1 = 0 then 1
else (n − 1) ∗ (if n − 2 = 0 then 1
else (n − 2) ∗ (if n-3=0 then 1
else . . .
```

$$
\begin{aligned}
F &= Y\ G \\
&=_\beta G\ (Y\ G) \\
&=_\beta G\ (Y\ G) \\
&=_\beta G\ (G\ (Y\ G)) \\
&\ldots
\end{aligned}
$$

Fakulteta

Fact = λfact.λn.if (IsZero n) C1 (Times n (fact (Pred n)))

Factorial = Y Fact

Factorial C2 = Y Fact C2

$=_\beta$ Fact (Y Fact) C2

$=_\beta$ (λfact.λn.if (IsZero n) C1 (Times n (fact (Pred n)))) (Y Fact) C2

$=_\beta$ (λn.if (IsZero n) C1 (Times n (Y Fact (Pred n)))) C2

$=_\beta$ if (IsZero C2) C1 (Times C2 (Y Fact (Pred C2)))

$=_\beta$ if False C1 (Times C2 (Y Fact C1)))

$=_\beta$ Times C2 (Y Fact C1)

= Times C2 (Factorial C1)

Ima vsak λ-izraz normalno obliko?

- Odgovor je ne!
- Let $L \equiv (\lambda x.xxy)(\lambda x.xxy)$: $L \rightarrow Ly \rightarrow Lyy \rightarrow \ldots$
- Naj bo $P \equiv (\lambda u.v)\ L$. P lahko reduciramo na dva načina.
  - $P \equiv (\lambda u.v)L \rightarrow ([L/u]v)L \equiv v$
  - $P \rightarrow (\lambda u.v)Ly$

    $\rightarrow (\lambda u.v)Lyy$

    $\rightarrow \ldots$
- P ima β-nf ampak tudi neskončno izpeljavo!
  - Λ-račun je neodločljiv jezik (parcialno izračunljiva funkcija).

Vrstni red evaluacije

- Nekatere λ-izraze lahko reduciramo na več načinov.
- Primer:
  1) $(\lambda x.(\lambda y.y\ x)\ z)\ v \rightarrow (\lambda y.y\ v)\ z \rightarrow z\ v$
  2) $(\lambda x.(\lambda y.y\ x)\ z)\ v \rightarrow (\lambda x.z\ x)\ v \rightarrow z\ v$
- Evaluacijske strategije:
  - Strategija normalne oblike
  - Klic po imenu
  - Klic po vrednosti

Evaluacijske strategije

Primer λ-izraza: (λx.x) ((λx.x) (λz. (λx.x) z))

Krajša oblika: id (id (λz.id z))

1) Polna β-redukcija je strategija, kjer lahko reduciramo poljuben redeks v vsaki točki izbire.

id (id (λz.<u>id z</u>))

→ id (<u>id (λz.z))</u>

→ <u>id (λz.z)</u>

→ λz.z

Evaluacijske strategije

2) Strategija *normalne oblike* izbere za redukcijo vedno skrajno levi, (najbolj) zunanji redeks.

$$
\begin{aligned}
& \underline{\text{id (id (}\lambda\text{z.id z))}} \\
\to\ & \underline{\text{id (}\lambda\text{z.id z)}} \\
\to\ & \lambda\text{z.}\underline{\text{id z}} \\
\to\ & \lambda\text{z.z}
\end{aligned}
$$

3) Strategija *klic po imenu* ne dovoli redukcij znotraj abstrakcij. Sicer je enaka strategiji normalne oblike.

$$
\begin{aligned}
& \underline{\text{id (id (}\lambda\text{z.id z))}} \\
\to\ & \underline{\text{id (}\lambda\text{z.id z)}} \\
\to\ & \lambda\text{z.}\underline{\text{id z}} \\
\to\ & /
\end{aligned}
$$

Evaluacijske strategije

4) Strategija *klic po vrednosti* vedno reducira samo (najbolj zunanji redeks, vendar po tem, ko so reducirani vsi redeksi na desni do vrednosti.

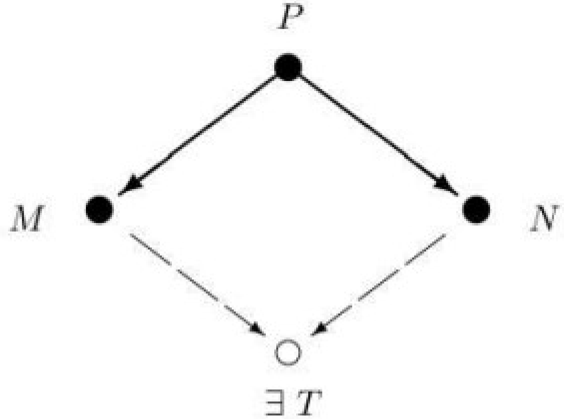
$$\begin{aligned} & \text{id } (\underline{\text{id } (\lambda z.\text{id } z)}) \\ \rightarrow\ & \underline{\text{id } (\lambda z.\text{id } z)} \\ \rightarrow\ & \lambda z.\text{id } z \\ \rightarrow\ & / \end{aligned}$$

V drugi vrstici argument (λz.id z) ni evaluiran pred celotnim izrazom, ker ni redeks.

Church-Rosserjev izrek

Centralni izrek λ-računa.

Izrek: Naj bo $P \twoheadrightarrow_\beta M$ in $P \twoheadrightarrow_\beta N$, potem obstaja T tako da $M \twoheadrightarrow_\beta T$ in $N \twoheadrightarrow_\beta T$.



Posledice CR izreka

- $M =_\beta N \Rightarrow \exists L{:}\ M \twoheadrightarrow_\beta L \land N \twoheadrightarrow_\beta L$
  - M je izpeljana iz N ⇒ imata isto vrednost!
- Če je N β-normalna oblika M potem $M \twoheadrightarrow_\beta N$
  - N je vrednost izraza M ⇒ obstaja izpeljava N iz M
- Vsak λ-izraz ima natančno eno β-nf
  - Konsistenca λ-računa: $\Lambda \nvdash \text{true} =_\beta \text{false}$

Nekatere lastnosti λ-računa

- Λ-račun je konsistenten jezik
- Λ-račun je ekvivalenten jezik Turingovem stroju
  - Alternativne definicije
    - Rekurzivno naštevan jezik (angl. r.e.): naštejemo lahko vse besede jezika (na pa tudi tistih, ki niso v jeziku)
    - Parcialno izračunljiva funkcija: ni definirana za vse argumente, matematični pogled
- λ-račun s tipi je totalna funkcija
  - Zelo omejen razred jezikov
- Bolj natančen opis totalnega Turingovega jezika ni znan!