## Lambda-kalkyyli

- Motivaatiota
- Lambda-termit, ekvivalenssi
- Muuttujat, substituutio, $\beta$-ekvivalenssi
- Laskenta, reduktiojärjestykset (scheme)
- Rekursio
- Ohjelmointirakenteen lambda-termeinä
- $\lambda$-kalkyyli vs. rekursiiviset funktiot



## Miksi $\lambda$-kalkyyli?

- Joukko-oppi 1800-luvun lopulla. Ongelmia:
  1. Russelin paradoksi ($\{x \mid x \notin x\}$ ?)
  2. Funktioiden staattisuus (lista pareja)
- Uusia yrityksiä formalisoida matematiikka:
  1. $\lambda$-kalkyyli (30-luvun alku): perustuu anonyymeillä funktioilla laskemiseen.
  2. Kategoriateoria (30–40-luku): keskittyy rakenteen kuvaamiseen ja funktioiden kompositionaalisuuteen.

  Gödel osoitti yritykset turhiksi.
- 50-luvulla keksittiin ottaa $\lambda$-kalkyyli ohjelmointikielen pohjaksi (Lisp)



## $\lambda$-termit

- Puhtaan $\lambda$-kalkyylin termi $M$ muodostetaan seuraavasti:

$$M := x \mid (M_1\ M_2) \mid (\lambda x.M).$$

- Tässä:
  - $x$ on *muuttuja*
  - $(M_1\ M_2)$ on *applikaatio*
  - $(\lambda x.M)$ on *abstraktio*
- Yleinen tapa on merkitä muuttujia symboleilla $f$, $g$, $x$, $y$ ja $z$ ja termejä $M$, $N$, $P$ ja $Q$.
- Sovelletussa $\lambda$-kalkyylissä käytetään lisäksi vakioita. Esim. numerot $(0, 1, \ldots, n)$ ja perusfunktiot $(+, -, \ldots)$



## Esimerkki

- Sovellettu termi $(\lambda x.(+\ x\ x)\ 5)$ tarkoittaa funktion $\lambda x.(+\ x\ x)$ käyttämistä vakioon 5.

$$(\lambda x.(+\ x\ x)\ 5) \Rightarrow (+\ 5\ 5) \Rightarrow 10$$

- $\lambda$-kalkyylin laskennassa korvataan aina monimutkainen termi yksinkertaisemmalla käyttäen symbolisia korvaussääntöjä, kunnes saadaan ulos termi, jolle ei voida tehdä enää mitään.

## Syntaktisia käytäntöjä

- Sulkeet jätetään pois termeistä, kun se on mahdollista. Tällöin:

$$(x\ y\ z) = ((x\ y)\ z)$$
$$\lambda xy.M = (\lambda x.(\lambda y.M))$$

- Traditionaalisesti käytetään seuraavia kolmea termiä "rakennuspalikoina"

$$I = \lambda x.x$$
$$K = \lambda xy.x$$
$$S = \lambda xyz.((x\ z)\ (y\ z))$$



## Vapaat ja sidotut muuttujat

- Termissä $M$ esiintyvä muuttuja $x$ on *sidottu*, mikäli termi on muotoa $M = \lambda x.M'$. Muut muuttujat ovat *vapaita*.
- Vapaat muuttujat voidaan määritellä myös rekursiivisesti:

$$free(x) = \{x\}$$
$$free(M\ N) = free(M) \cup free(N)$$
$$free(\lambda x.M) = free(M) \setminus \{x\}$$

- $\lambda$-termiä, jossa ei ole vapaita muuttujia kutsutaan *kombinaattoriksi*.



## $\alpha$-, $\beta$-, ja $\eta$-reduktiot (1/2)

- $\alpha$-reduktiossa vaihdetaan $\lambda$-termissä esiintyvä muuttuja toiseksi:

$$\lambda x.xx\ [z/x] \Rightarrow_\alpha \lambda z.zz$$

- $\beta$-reduktiossa käytetään $\lambda$-termiä jollekin argumentille:

$$(\lambda x.xy)\ (w\ z) \Rightarrow_\beta (w\ z)y$$

(Scheme-funktiot perustuvat $\beta$-reduktioon).
Formaalisti $\beta$-sääntö esitetään seuraavasti:

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



## $\alpha$-, $\beta$-, ja $\eta$-reduktiot (2/2)

- Muotoa $(\lambda x.M)\ N$ olevaa termiä kutsutaan $\beta$-*reduktiolausekkeeksi* ($\beta$-redex). Mikäli termissä ei ole yhtään $\beta$-redex alitermiä, se on $\beta$-normaalimuodossa.
- Kolmas reduktiomuoto on $\eta$-reduktio:

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

## Muuttujien korvaaminen (1/2)

- Tarkastellaan termiä:

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

- Mikäli nyt käytettäisiin $\beta$-sääntöä kuten yllä, saataisiin ulos termi:

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

- Ongelmana on, että ensimmäinen $y$ muuttui juuri vapaasta sidotuksi, mikä muuttaa termin merkityksen.
- Ongelmasta selvitään, kun määritellään korvausoperaattori [] siten, että se nimeää uudelleen sidotut muuttujat tarvittaessa käyttäen $\alpha$-sääntöä.



## Muuttujien korvaaminen (2/2)

- Formaalisti substituutio-operaattori määritellään tapauksittain:

$$\begin{aligned}
x\ [N/x] &= N\\
y\ [N/x] &= y\\
(P\ Q)[N/x] &= (P[N/x]\ Q[N/x])\\
(\lambda x.P)[N/x] &= \lambda x.P\\
(\lambda y.P)[N/x] &= \lambda y.P[N/x], \quad y \neq x, y \notin free(N)\\
(\lambda y.P)[N/x] &= \lambda z.P[y/z][N/x], \quad y \neq x, y \in free(N)
\end{aligned}$$



## Esimerkki

- Sievennetään lauseke:

$$\begin{aligned}
(\lambda x.\lambda y.(x\ y))\ (\lambda z.(z\ y)) &\Rightarrow (\lambda y.(x\ y))[(\lambda z.(z\ y))/x]\\
&\Rightarrow (\lambda y.(x\ y))[y'/y][(\lambda z.(z\ y))/x]\\
&\Rightarrow (\lambda y'.(x\ y'))[(\lambda z.(z\ y))/x]\\
&\Rightarrow (\lambda y'.((\lambda z.(z\ y))\ y'))\\
&\Rightarrow (\lambda y'.((z\ y))[y'/z])\\
&\Rightarrow \lambda y'.(y'\ y)
\end{aligned}$$



## Reduktiosäännöt

- Puhtaassa $\lambda$-kalkyylissä voidaan käyttää $\alpha$- ja $\beta$-reduktioiden lisäksi myös seuraavia johtosääntöjä:

$$\begin{aligned}
M = N &\Rightarrow N = M && \text{(kommutatiivisuus)}\\
M = N \wedge N = P &\Rightarrow M = P && \text{(transitiivisuus)}\\
M = M' \wedge N = N' &\Rightarrow MN = M'N' && \text{(kongruenssi 1.)}\\
M = M' &\Rightarrow \lambda x.M = \lambda x.M' && \text{(kongruenssi 2.)}
\end{aligned}$$

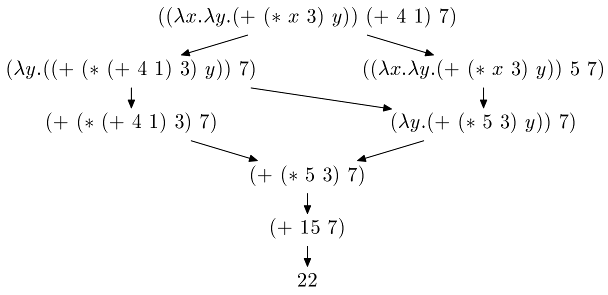
- Sovelletussa kalkyylissä lisätään mukaan säännöt vakioiden redusoinnista. (esim. $5 + 5 \Rightarrow 10$)

## Laskenta (1/2)

- $\lambda$-termi $M = (\lambda x.\lambda y.(+\ (*\ x\ 3)\ y))\ (+\ 4\ 1)\ 7)$ voidaan redusoida kahdella tapaa:
  1. $M \Rightarrow (\lambda y.((+\ (*\ (+\ 4\ 1)\ 3)\ y))\ 7)$
  2. $M \Rightarrow ((\lambda x.\lambda y.(+\ (*\ x\ 3)\ y))\ 5\ 7)$
- Näistä ensimmäistä kutsutaan *normaaliksi* reduktioksi (normal-order, call-by-name, $\lambda$-value) ja toista *applikatiiviseksi* reduktioksi (applicative-order, call-by-value)
- Applikatiivisessa järjestyksessä redusoidaan funktion argumentit ennen funktion kutsumista, normaalijärjestyksessä argumentit annetaan sellaisenaan. Scheme-tulkit käyttävät yleensä applikatiivista.



## Esimerkki






## Laskenta (2/2)

- On mahdollista, että $\lambda$-termillä ei ole $\beta$-normaalimuotoa:

$$(\lambda x.xx\ \lambda x.xx) \Rightarrow (\lambda x.xx\ \lambda x.xx)$$

- Mikäli termillä on $\beta$-normaalimuoto, se on yksikäsitteinen.
- Normaalijärjestys löytää normaalimuodon aina, kun se on olemassa, applikatiivinen ei välttämättä löydä. Esim.

$$((\lambda xy.x)\ z\ (\lambda z.zz\ \lambda z.zz)) \Rightarrow z,$$

mutta applikatiivinen jää ikuiseen silmukkaan redusoimaan jälkimmäistä argumenttia.



## Rekursio (1/3)

- Puhtaassa $\lambda$-kalkyylissä ei ole `define`- tai `letrec`-erikoismuotoja. Miten siis rekursio?
- Muutetaan funktio:

```
(define (factorial n)
  (if (zero? n) 1 (* n (factorial (- n 1)))))
```

puhtaaksi $\lambda$-termiksi.

- Aluksi muutetaan funktio muotoon:

```
(define factorial
  (lambda (n) (if (zero? n)
                  1
                  (* n (factorial (- n 1))))))
```

## Rekursio (2/3)

- Seuraavaksi korvataan muuttuja `factorial` sopivalla $\lambda$-termillä:

```
(lambda (g)
  (lambda (n) (if (zero? n)
                  1
                  (* n (g (- n 1))))))
```

- Apuun tarvitaan vielä *kiintopistekombinaattoria* $Y$, jolle pätee:

$$(Y\ F) = (F\ (Y\ F))$$

- Sopiva kombinaattori voisi olla esim:

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



## Rekursio (3/3)

- Valitettavasti ylläoleva kombinaattori ei toimi kuin normaalijärjestyksellä, joten tarvitaan kombinaattori $Y'$:

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

- Rekursiivinen kertomafunktio saadaan nyt seuraavasti:

```
(Y' (lambda (g)
      (lambda (n) (if (zero? n)
                      1
                      (* n (g (- n 1))))))
```



## Ohjelmointikielen rakenteet $\lambda$-termeinä

- Totuusarvot: $\mathbf{T} = \lambda xy.x$ $\quad \mathbf{F} = \lambda xy.y$
- if $B$ then $M$ else $N$: $BMN$
- konnektiivit: not : $\lambda x.\mathbf{FT}$ , or: $\lambda xy.x\mathbf{T}y$
- cons($M$, $N$): $\lambda z.zMN = [M, N]$ $\quad$ car(M): $M\mathbf{T}$ $\quad$ cdr(M): $M\mathbf{F}$.
- Numeraalit:

$$\begin{aligned} 0 =& I = \lambda x.x \\ s(x) =& \lambda x.[F, x] \\ s^{-1}(x) =& \lambda x.x\mathbf{F} \\ zero(x) =& \lambda x.x\mathbf{T} \end{aligned}$$



## $\lambda$-kalkyyli ja rekursiiviset funktiot

- Tiedetään, että $\mu$-rekursiivisiset funktiot ovat formalismina ekvivalentit Turingin koneiden kanssa.
- $\mu$-rekursiiviset funktiot saadaan nollan, seuraajafunktion, projektion, komposition primitiivirekursion ja rajoittamattoman minimoinnin avulla.
- Nolla ja seuraaja olivat edellisellä kalvolla. Projektio:

$$\pi_i^n = \lambda x_1 \ldots x_n.x_i$$

Kompositio ja primitiivirekursio triviaaleja.
- Rajoittamaton minimointi saadaan scheme-funktiolla:

```
(define (minimize P i)
  (if (P i) i
          (minimize P (+ i 1))))
```

$\Longrightarrow$ $\lambda$-kalkyyli ja Turingin koneet yhtä vahvoja.

## $Y$-kombinaattorin reduktio

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

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