



## Dagens tema

# $\lambda$ -KALKYLE

1/15

Forelesning 12 – 12.11.2003

## Motivasjon

Hvis man skal forstå programmeringsspråk, så man forstå hva det vil si å beregne noe.

*Hva er det å beregne noe?*

Historie: Church, Turing m.fl. på 30-tallet. Hva er en beregning? Hva kan beregnes? Hva er beregnbart?

Forelesning 12 – 12.11.2003

2/15

## Lambdakalkyle

Intendert som et fundament for all matematikk.

Opprinnelig en teori om funksjoner.

To syn på funksjoner

- en input/output-korrelasjon, matematisk
- en regel, *hvordan* input blir til output

$\lambda$ -kalkyle er en presis formalisering av nummer to!

Forelesning 12 – 12.11.2003

3/15

## Lambdakalkyle

Veldig abstrakt - og det er det som er fint!

Vi tar bort det partikulære og sitter igjen med ideen om en funksjon - hva en funksjon er.

Dette gjør at vi får en modell for beregninger; en forklaring av hva beregninger er.

**Svar på spørsmålet: det som kan beregnes er nøyaktig de funksjonene man kan uttrykke i lambdakalkyle.**

Beregnbarhet - som det som kan beregnes av en turingmaskin.

Kraften i dette: med bare dette enkle språket så kan vi gjøre alt vi overhodet kan beregne!

Forelesning 12 – 12.11.2003

4/15

## Lambdakalkyle

Senere: mange utvidelser, typet/utypet, omskriving, innflytelse på programmeringsspråk.

En felles basis for funksjonelle programmeringsspråk, nettopp fordi vi har tatt bort det partikulære.

Viktigheten av semantikk: hva er semantikken til et program?  
Kanskje viktigere: hva er semantikken til en algoritme?

## Utypet lambdakalkyle

- ingen typer (ulikt ML, som er sterkt typet!)
- alle uttrykk, tolket som funksjoner, kan anvendes på alle uttrykk

### Eksempel - identitetsfunksjonen:

Scheme: `(lambda (x) (x))`

ML: `fn x => x`

$\lambda$ -kalkyle:  $\lambda x.x$

Scheme er ikke sterkt typet; "alt kan anvendes på alt".

## Utypet lambdakalkyle - først uformelt

Det fins to grunnleggende operasjoner i  $\lambda$ -kalkyle, *applikasjon* og *abstraksjon*.

### Applikasjon

Hvis  $F$  og  $G$  er  $\lambda$ -uttrykk, så er  $FG$  en *applikasjon*; vi ser på  $F$  som en funksjon og  $G$  som argumentet til denne funksjonen.

### Abstraksjon

Hvis  $F[x]$  er et uttrykk hvor  $x$  forekommer, så svarer  $\lambda x.F[x]$  til funksjonen som tar  $x$  som argument og gir  $M[x]$  som resultat.

## Utypet lambdakalkyle - først uformelt

Eksempel - abstraksjon og applikasjon:

Fra  $x^2 + 3$  *abstraherer* vi og får  $\lambda x.x^2 + 3$ . Dette uttrykket kan så *appliseres* på et tall (og resultatet regnes ut) slik:

$$\begin{aligned} & (\lambda x.x^2 + 3)4 \\ & \rightsquigarrow 4^2 + 3 \\ & \rightsquigarrow 19 \end{aligned}$$

Den første overgangen kalles en  $\beta$ -konversjon. Se likheten med ML!

## Utypet lambdakalkyle

Vi antar at en uendelig mengde *variable*,  $x, y, z, \dots$ , er gitt.

Alle syntaktiske objekter er **termer**. Definisjon:

1. En variabel er en term. (Basistilfellet.)
2. Hvis  $x$  er en variabel og  $F$  er en term, så er  $(\lambda x.F)$  en term.
3. Hvis  $F$  og  $G$  er termer, så er  $(FG)$  en term.

Formulert som en grammatikk:

$$\langle term \rangle \rightarrow \langle var \rangle \mid \langle (\lambda \langle var \rangle . \langle term \rangle) \rangle \mid \langle ( \langle term \rangle \langle term \rangle ) \rangle$$

## Utypet lambdakalkyle

Eksempler på  $\lambda$ -termer:

$x$   
 $(xy)$   
 $(\lambda x.x)$   
 $((\lambda x.x)y)$   
 $(\lambda x.(xx))$   
 $((\lambda x.(xx))(\lambda x.(xx)))$   
 $(\lambda x.(\lambda y.(xy)))$

## Utypet lambdakalkyle

Noen parenteskonvensjoner:

$FGHI$  er en forkortelse for  $((((FG)H)I))$ . (Tenk ML. . .)

$\lambda xyx.F$  er en forkortelse for  $(\lambda x.(\lambda y.(\lambda z.F)))$

Fra forrige foil:

$x$	$x$
$(xy)$	$xy$
$(\lambda x.x)$	$\lambda x.x$
$((\lambda x.x)y)$	$(\lambda x.x)y$
$(\lambda x.(xx))$	$\lambda x.xx$
$((\lambda x.(xx))(\lambda x.(xx)))$	$(\lambda x.xx)(\lambda x.xx)$
$(\lambda x.(\lambda y.(xy)))$	$\lambda xy.xy$

## Utypet lambdakalkyle

### Frie og bundne variable:

- $FV(x) = \{x\}$
- $FV(FG) = FV(F) \cup FV(G)$
- $FV(\lambda x.F) = FV(F) \setminus \{x\}$

De frie variable i uttrykket  $G$  er gitt ved  $FV(G)$ .

Intuitivt: I  $\lambda x.F$  bindes variabelen  $x$  og skopet til bindingen er  $F$ .

### Substitusjon:

$F[x/G]$  betyr uttrykket  $F$  hvor alle *frie* forekomster av  $x$  er erstattet med  $G$ .

## Utypet lambdakalkyle

$\beta$ -konversjon:

$$(\lambda x.F)G \rightsquigarrow F[x/G]$$

Dette er den *syntaktiske* versjonen av at en funksjonen blir gitt et argument som input. . .

I tillegg har vi  $\alpha$ -konversjon:  $\lambda x.F \rightsquigarrow \lambda y.(F[x/y])$

## Utypet lambdakalkyle

Eksempel på definerbarhet. Church-numeraler.

La  $F^n G$  bety  $F(F(\dots(FG)))$  med  $n$  antall  $F$ 'er.

Vi kan da *definere* naturlige tall på følgende måte:

$$\begin{aligned} c_0 &= \lambda f x.x \\ c_1 &= \lambda f x.fx \\ c_2 &= \lambda f x.f(fx) \end{aligned}$$

Generelt  $c_n$  er definert som  $\lambda x.f^n(x)$  og står for det naturlige tallet  $n$ .

Oppgave(vanskelig!): Hvis naturlige tall representeres på denne måten, hvordan kan addisjon og multiplikasjon defineres?

## Utypet lambdakalkyle

Reduksjoner og normalform.

$\beta$ -konversjoner kan anvendes til å *redusere*  $\lambda$ -termer. Når en  $\lambda$ -term ikke kan *reduseres* lenger, sier vi at den er på *normalform*.

I forskjellige varianter av  $\lambda$ -kalkyle viser man ofte at man alltid kan redusere en term til dens *unike* normalform. På en måte kan vi se på denne som *verdien* til funksjonen.

Prosessen som leder frem til normalformen kalles for *normalisering*.