Dagens tema

λ-KALKYLE

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?

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

λ-kalkyle er en presis formalisering av nummer to!

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!

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`
λ-kalkyle: $\lambda x.x$

Scheme er ikke sterkt typet; “alt kan anvendes på alt”.

Utypet lambdakalkyle - først uformelt

Det fins to grunnleggende operasjoner i λ-kalkyle, *applikasjon* og *abstraksjon*.

Applikasjon

Hvis F og G er λ-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:

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

Den første overgangen kalles en β-konversjon.
Se likheten med ML!

Utypet lambdakalkyle

Vi antar at en uendelig mengde *variable*, $x, y, z, \ldots$, 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:

$$\langle \mathit{term} \rangle \rightarrow \langle \mathit{var} \rangle \mid \mathbf{(}\langle \mathit{term} \rangle\langle \mathit{term} \rangle\mathbf{)} \mid \mathbf{(}\lambda\ \langle \mathit{var} \rangle\mathbf{.}\langle \mathit{term} \rangle\mathbf{)}$$

Utypet lambdakalkyle

Eksempler på λ-termer:

x
(xy)
(λx.x)
((λx.x)y)
(λx.(xx))
((λx.(xx))(λx.(xx)))
(λx.(λy.(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
(λx.x)	λx.x
((λx.x)y)	(λx.x)y
(λx.(xx))	λx.xx
((λx.(xx))(λx.(xx)))	(λx.xx)(λx.xx)
(λx.(λy.(xy)))	λxy.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* variablen 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

β-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 α-konversjon: $\lambda x.F \rightsquigarrow \lambda y.(F[x/y])$

Utypet lambdakalkyle

Eksempel på definerbarhet. Church-numeraler.

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

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

$c_0 = \lambda fx.x$
$c_1 = \lambda fx.fx$
$c_2 = \lambda fx.f(fx)$

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.

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

I forskjellige varianter av λ-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*.