Fra LAMBDA-KALKULER til funktionelle sprog

Grundlaget for funktionelle programmeringssprog blev skabt af en matematiker, før computeren var opfundet.

Af Dan Mygind
[prosabladet@prosa.dk]

Alonzo Church skulle løse et problem; et matematisk problem kaldet Entscheidungsproblem, eller beslutningsproblemet på dansk. Er det muligt at konstruere en algoritme, der kan tage ethvert matematisk udsagn og bevise, at det enten er sandt eller falsk?

For at kunne besvare det spørgsmål måtte Alonzo Church først definere et formelt system i matematisk logik, der kunne hjælpe ham med at modellere beregnelighed. Lambda-kalkuler var født, og det hjalp Alonzo Church til at løse Entscheidungsproblemet i 1936. Han kom frem til, at det ikke er muligt at konstruere en algoritme, der kan bevise ethvert matematisk udsagn. Kort tid efter nåede en anden teoretisk matematiker, Alan Turing, frem til samme konklusion som Alonzo; ikke ved at benytte lambda-kalkuler, men ved hjælp af sin logiske konstruktion, Turing-maskine.

Det første funktionelle sprog

På det tidspunkt fandtes der slet ikke nogle computere, men Alonzo Churchs og Alan Turings arbejde fik vidtrækkende betydning for de computere og den software, som blev bygget i de følgende år. Turing-maskinen blev det teoretiske fundament for de imperative sprog, mens Alonzo Churchs lambda-kalkuler blev grundlaget for funktionelle sprog. Den amerikanske datalog og pioner inden for kunstig intelligens, John McCarthy, var den første til at blive inspireret af lambda-kalkuler til at udvikle et programmeringssprog. Han så muligheden for at udnytte matematisk logik til at skabe kunstig intelligens og udviklede Lisp i 1958 på basis af Churchs lambda-kalkuler. Lisp blev hurtigt det foretrukne programmeringssprog for datidens kunstig intelligens-programmer. Lisp var blot det første i en række funktionelle programmeringssprog, der siden er kommet til. Mange af dem anvendes mest til forskning og undervisning, men i de senere år er der kommet nye sprog til, som bygger videre på de funktionelle dyder og kombinerer dem med objektorienterede principper. Resultatet er hybride sprog som F# på .Net-platformen og Scala til Java-platformen, der er anvendes til at udvikle kommercielle systemer. Til forskel fra de hybride former findes eksempelvis Haskell, der betegnes som et rent (pure) funktionelt programmeringssprog. Haskell tillader eksempelvis ikke, at der er nogen sideeffekter, når en funktion eksekveres, til forskel fra F#.

Rene og hybride former

Men det er ikke kun sprog som F#, der baserer sig på det funktionelle paradigme og har fået tilsat objektorienterede features, som er med til at bringe funktionel programmering ud til mainstream-programmører. Funktionelle features begynder at snige sig ind i eksisterende mainstream-objektorienterede programmeringssprog som eksempelvis C#. Specielt udbredelsen af flerkerne-processorer er med til at gøre det funktionelle programmeringsparadigme interessant. I stedet for at øge klok-frekvensen tilbyder chip-producenterne i dag mere regnekraft ved at introducere flere kerner på en chip. Det er således op til programmørerne at udnytte kernerne ved at skrive kode, der kan deles op og eksekveres parallelt. Her kan funktionelle programmeringssprog med immutable objekter (objekter, som ikke kan ændres, efter at de er oprettet), ingen sideeffekter og ingen delt memory mellem funktioner være med til at gøre parallel eksekvering af kode på flere kerner nemmere.

Så da Alonzo Church i 1936 løste Entscheidungsproblemet ved hjælp af lambda-kalkulerne, var han på en måde også med til at løse problemer for programmører i dag.

Alonzo Churchs lambda-kalkuler inspirerede den amerikanske datalog og AI-pioner John McCarthy til at udvikle Lisp.
(Foto: null0 Licensed under CC BY-SA 2.0)

Lambda-kalkule

Lambda-kalkule er navngivet efter det græske bogstav λ. En funktion, som lægger 1 til et tal, skrives som λx. x+1.
Læs mere om lambdakalkuler her http://goo.gl/NQfCs

Imperative, deklarative, objektorienterede og funktionelle sprog

Imperative sprog beskriver, hvordan computeren skal foretage udregninger. De baserer sig på tilstande og en sekvens af operationer, der ændrer på tilstandene. Det kan eksempelvis være et loop, der løbende summerer værdien af elementerne i et array. Her ændres værdien (tilstanden) af variablene s og i :

```
i := 0;
s := 0;
while i < length(A)
{
  s := s+A[i];
  i := i+1
}
```

Objektorienterede sprog er imperative, da objekterne i et kørende program løbende ændrer tilstand.

Funktionelle sprog hører til de deklarative sprog, hvor man beskriver, hvad computeren skal nå frem til. Du anvender formentlig allerede et deklarativt sprog som SQL. Hvis du ønsker at finde alle personer over 18 år i din database, beskriver du blot i SQL, hvad du ønsker:

```
Select name From Persons where age > 18
```

Funktionelle Sprog

Der findes et væld af eksperimentale funktionelle sprog. Her er nogle af de mere kendte og udbredte sammen med deres oprindelsesår.

Lisp	1958
ML	1973
Scheme	1975
Caml	1985
Erlang	1986
Haskell	1990
oCaml	1996
Scala	2004
F#	2005
Clojure	2009
Elixir	2012
Elm	2012

Ramón Soto Mathiesen blev bidt af funktionelle sprog på Københavns Universitet. Han opfordrer flere udviklere til at prøve de funktionelle sprogs muligheder. (Fotograf: Bobby Anwar)

Hvorfor prøver du ikke et FUNKTIONELT SPROG?

Funktionelle programmeringssprog er matematiknære. Derfor mener systemudvikler og teknisk chef Ramón Soto Mathiesen, at flere bør prøve at udvikle i funktionelle sprog.

Af Dan Mygind
[prosabladet@prosa.dk]

– Når du udstikker tekniske retningslinjer for et projekt, er det rart at vide, at det valgte programmeringssprog gør produktet mere robust.

Ramón Soto Mathiesen er ikke i tvivl om fordelen ved at anvende funktionelle sprog. Han anvender selv det funktionelle programmeringssprog F# til daglig, og som CTO hos Delegate har han lidt mere ro i maven, når selskabets CRM-installations- og deployment-værktøj, DAXIF#, hovedsageligt er kodet i F#.

– Funktionelle programmeringssprog er tættere på matematikken, så det er nemmere at teste og bevise, at koden virker, argumenterer Ramón Soto Mathiesen.

Han har en lang baggrund inden for it og har kodet i mainstream-sprog som C#, C++ og Javascript, men det er funktionelle sprog som F#, der tænder passionen i den dansk-spanske datalog. Ifølge Ramón Soto Mathiesens erfaringer bliver kodebasen mindre og mere overskuelig, ligesom det er nemmere at udnytte multikerneprocessorer, da funktionelle sprog er velegnede til parallelprogrammering og håndtering af samtidighed. Samtidig speedes udviklingstiden også op.

– Udvikling i C# og Java og lignende objektorienterede sprog tager længere tid end med unktionelle sprog, mener Ramón Soto Mathiesen.

Fra objektorientering til funktionel

Udviklingstiden afhænger selvfølgelig af, hvor godt udviklerne kender det pågældende programmeringssprog og det bagvedliggende programmeringsparadigme. Langt de fleste udviklere i dag er fortrolige med den objektorienterede tankegang, mens det er færre, der er fortrolige med den funktionelle tilgang. Ramón Soto Mathiesen har selv erfaret, at det ikke altid er nemt at få udviklere, som er fortrolige med det objektorienterede paradigme, til uden videre at skifte til et funktionelt sprog.

– Medarbejderne var i starten ikke så glade for det, siger han om tiden for tre år siden, da han forsøgte at indføre F# som udviklingssproget for DAXIF#.

Derfor blev værktøjerne, der gør installation og udrulning af Microsofts Dynamics CRM nemmere, i første omgang udviklet i C# og først senere skrevet i F#. Ved at skrive små kompakte F#-moduler i stedet for at anvende >>

”Funktionelle programmeringssprog er tættere på matematikken, så det er nemmere at teste og bevise, at koden virker”

Ramón Soto Mathiesen, systemudvikler og teknisk chef

C#-klasser blev antallet af kodelinjer reduceret til omkring en trediedel af den oprindelige kodebase.

Ramón Soto Mathiesen fremhæver blandt andet F#'s REPL (Read, Evaluate, Print Loop), der er med til at gøre udviklingen hurtigere.

– Med F# kan en funktion evalueres 'on the fly', og du kan se, at den lille komponent virker efter hensigten, uden at du skal rekompilere hele dit projekt. Det er en stor fordel, da en lille kodeændring ikke kræver en ny timelang build af hele projektet efterfulgt af unit-tests, som jeg har oplevet på andre projekter, siger han.

Funktionel afsmitning

I dag har C# også REPL, men den er ifølge Ramón Soto Mathiesen ikke på niveau med F#'s, der var født med REPL. Det er dog et eksempel på, hvordan funktionelle sprogs features og værktøjer langsomt trænger ind på imperative sprog. Et andet eksempel er LINQ (Language-integrated Query), der blev del af C# 3.0 i slutningen af 2007. Det lille forespørgselssprog med rødder i funktionelle sprog har gjort mange C#-udviklere verden over meget glade.

– Spørger du folk, hvorfor de kan lide C#, siger de tit, at det er på grund af LINQ. Når man så spørger, om de vil prøve F#, siger de nej. Hvorfor skulle jeg gøre det? Men hvis de godt kan lide LINQ, hvorfor så ikke kode i et funktionsorienteret sprog som F# hele tiden?, lyder det retorisk fra Ramón Soto Mathiesen.

Han er da også selv meget involveret i at udbrede kendskabet til de funktionelle sprogs kvaliteter. Han var med til at starte gruppen Funktionelle Københavnere, der mødes minimum en gang om måneden til præsentationer og efterfølgende diskussioner om funktionelle sprog. Han arbejder også sammen med kursushuset Skills House om at undervise udviklere i F#.

Ramón Soto Mathiesen understreger dog, at han ikke er blind for andre programmeringsparadigmers fordele. Som enhver god udvikler ved, handler det om at anvende de værktøjer, som er bedst egnede til en given opgave.

– Eksempelvis anvender vi i stor stil TypeScript, som slog FunScript (F# til klientsiden, red.) ud af vores palet af teknologier. Så vi laver ikke kun funktionsprogrammering for pedanteriets skyld. Nej, vi anvender de værktøjer, som giver mest mening uanset paradigme, siger han.

Let for matematikere

Det var på Datalogisk Institut på Københavns Universitet, at han fik øjnene op for mulighederne med funktionelle sprog. Danmark har generelt gode undervisere i funktionelle sprog på universitetsuddannelserne, men det kniber for erhvervslivet at tage funktionel programmering i anvendelse. Efter universitetet oplevede Ramón Soto Mathiesen, at det nærmest var forbudt at anvende funktionelle programmeringssprog, fordi folk i ledende positioner, efter hans mening, ikke forstår det så godt.

Der er dog virksomheder, især inden for den finansielle sektor, som kan se idéen med funktionelle sprog. Eksempelvis anvender PFA's aktuarer F# ligesom virksomheden Simcorp, der leverer finansielle værktøjer, anvender oCaml og F#.

– Folk med en matematisk baggrund kan godt se pointen med F# og andre funktionelle sprog; det minder meget om dengang, de definerede matematiske funktioner, siger han og nævner aktuaren Kristian Schmidt hos PFA, der tog F# til sig og begyndte at undervise de andre aktuarer i at anvende programmeringssproget.

Selvom man ikke har en matematisk baggrund, kan man godt lære F#, men det kan tage lidt tid at vænne sig til det funktionelle paradigme.

– Jeg har prøvet at undervise udviklere uden akademisk baggrund i F# et par gange. En dags undervisning var ikke helt nok. Det skal dog ikke afskrække folk. Det er absolut værd at bruge lidt tid på at lære funktionelle sprog, slår Ramón Soto Mathiesen fast.

Funktionelle Københavnere

Sammen med sin tidligere medstuderende, Joakim Ahnfelt-Rønne, har Ramón Soto Mathiesen etableret mødegruppen Funktionelle Københavnere. Der afholdes jævnligt møder i PROSA med talere fra ind- og udland samt diskussioner om funktionelle sprog som F#, Erlang, oCaml og andre. Blandt talerne har været Don Symes, manden bag F#. Læs mere her: http://www.meetup.com/MoedegruppeFunktionelleKoebenhavnere