Funktionell programmeringsparadigm

Funktioner
Listor

Funktioner är grunden

- Program - en avbildning från invärden till utvärden
- Karakteriseras av användningen av funktioner och uttryck
- Sammansättningar av flera enklare funktioner
- Inga sidoeffekter, kommandon eller variabler
- Programmen är skrivna helt genom användning av uttryck, funktioner och deklarationer
- Kraftfulla begrepp, som högre ordningens funktioner och lat evaluering
- Exempel på språk
  - SML, Haskell, Lisp, Scheme, Miranda, SISAL

Varför inte kraftfulla kommandon?

- Mer komplicerad semantik
- Sidoeffekter fungerar dåligt med lat evaluering
- Antingen:
  - Helt ta bort variabler (Miranda, Haskell)
  - Begränsa funktionalismen i språket och tillåta variabler (ML)

Gemensamma begrepp

- Enkel och kompakt syntax
- Funktionsbegreppet, är mycket centralt
- Uttrycksorienterat
- Rekursion
- Mönstermatchning
- Deklaration av funktioner ses som fastställande av egenskaper hos funktioner snarare än ett sätt att beräkna en funktion
- Högre ordningens funktioner
- Automatisk skräphantering
- Enkel och klar semantik

Egenskaper

- Egendefinierade datatyper
- Statisk räckviddsbindning
- Statisk typkontroll
- Automatisk typinferens
- Parametrisk polymorfism
- Överlagring
- Parallellitet

Rena funktionella språk

- Lat evaluering
- Inga imperativa inslag
  - Inga sidoeffekter

Speciella egenskaper hos vissa språk

- Modulsystem
  - ML har ett mycket avancerat modulsystem
- Överlagring
  - Haskells typklasser

Mönstermatchning

- Case-uttryck
- Funktionsdefinitioner
- Kort och kompakt
- Tydligt
- Kopplat till matematisk notation

Värden och typer

- Samma typer som de flesta språk
  - Ingen selektiv uppdatering
  - Eftersom det inte finns uppdatering av variabler kan olika värden dela gemensamma delar
- Värden läggs på högen
  - En implicit deallokerare måste existera
  - Programmeraren behöver inte bry sig om minneshanteringen
- Rekursiva typer till exempel listor och träd kan hanteras effektivt i funktionella språk
- Arrayer är mycket svårare

Listor

- Den viktigaste typen av sammansatta värden i funktionell programmering
- Inbyggda konstruktorer och funktioner
- Enkelt att definiera egna funktioner, oftast med hjälp av rekursion

Listor

```
- datatype 'a list = nil | cons of 'a * 'a list;
- x :: xs = cons(x,xs)
- [a,...,z] = a :: ... :: z :: nil
```

- List comprehension

```
- ints = [1,2...]
- ints = [n+1 | n <- ints]
- [n | n <- ints; n mod 2 = 0]
```

Högre ordningens funktioner

- Funktioner är första klassens värden i funktionella språk
- Tar en funktion som argument och/eller ger en funktion som resultat
  - Viktig del i funktionell programmering
  - Representera data på ett mer uttrycksfullt sätt
  - Kan användas för att skriva generella funktioner

```
fun power n b = if n=0 then 1.0 else b*power(n-1) b
val sqr = power 2
and cube = power 3
```

Partiellt applicerbara funktioner

- filter, map

```
fun reduce f u =
   let fun g nil = u
         | g (x::xs) = f x (g xs)
   in g end
val sum = reduce op+ 0
and product = reduce op* 1
and implode = reduce op^ ""
```

Lat evaluering

- Argumentet evalueras när det behövs istället för vid anrop, men endast så mycket som behövs
  - Funktioner kan returnera partiellt beräknade resultat
- Oändliga listor

```
fun from n = n :: from (n+1)
```

- Programmeraren har liten kontroll över evalueringsordningen
- Fungerar mycket dåligt tillsammans med sidoeffekter
  - Sidoeffekter endast i små noggrant skrivna moduler

Modellera tillstånd

- Görs indirekt genom en funktion med objektets aktuella värde som argument
- Passar dåligt till ickedeterministiska applikationer
- Upplevs ofta onaturligt

Pragmatik

- Transformation och bevis
- Effektivitet
  - Förr 100 ggr långsammare, nu 3-5 jfrt med imperativa språk
- Parallellitet