Programmering og Problemløsning

Typer og Mønstergenkendelse (Del 1)

Martin Elsman

Datalogisk Institut, Københavns Universitet (DIKU)

- Typer
- Typeforkortelser
- Record typer (generaliserede produkter)

Typer

Typer kan forstås som mængder af værdier.

- Typebegrebet giver os således et sprog for at klassificere værdier.
- Vi kan for eksempel tale om at en funktion returnerer et heltal (type `int`).
- F# oversætteren kan "type-tjekke" vores programmer for at sikre os mod en lang række fejl når programmet køres!

Nogle emner vi vil dække:

- Hvordan kan type-begrebet udvides til at klassificere flere slags værdier, såsom funktioner (og funktioner der returnerer funktioner).
- Hvordan kan vi skrive genbrugelige *type-generiske* funktioner, der kan køre på data af forskellig type.
- Vi vil senere se hvordan type-begrebet kan udvides til at kunne beskrive træer, grafer og andre strukturelle datastrukturer.

Typer kan forstås som mængder af værdier

Eksempler:

`int`	$\approx$	$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
`float`	$\approx$	$\mathbb{R}$
`int * float`	$\approx$	$\mathbb{Z} \times \mathbb{R} = \{\ldots, (3, 1.2), \ldots\}$ (sæt af alle par med elementer fra $\mathbb{Z}$ og $\mathbb{R}$)
`int list`	$\approx$	$\{[], [1], [2], \ldots, [1; -2], \ldots\}$
`int -> float`	$\approx$	$\mathbb{Z} \to \mathbb{R}$ (sæt af alle afbildninger fra $\mathbb{Z}$ til $\mathbb{R}$)
`bool`	=	$\{$`true`,`false`$\}$
`unit`	=	$\{()\}$

Spørgsmål:

1. Hvorfor $\approx$ for de første fire tilfælde?
2. Hvorfor $\approx$ for funktioner?

Typeforkortelser

Det er nemt i F# at give et navn til en type for derved at gøre kode lettere læselig.

```
type department = string
type costs = (department * float) list
let total (costs:costs) : float =
  List.fold (fun acc (_,f) -> acc+f) 0.0 costs
```

Bemærk:

- Typen `department` er blot et synonym for typen `string`.
- Funktionen `total` kan derfor benyttes på alle værdier af typen `(string*float) list`.

Type-generiske type-forkortelser

Type-forkortelser kan være generiske således at det er muligt at skrive generisk kode der henviser til en type-forkortelse:

```
// association lists mapping strings to values of type 'a
type 'a alist = (string * 'a) list
let add (m:'a alist) (s:string) (v:'a) : 'a alist =
   (s,v)::m
let rec look (m:'a alist) (s:string) : 'a option =
  match m with
     [] -> None
   | h::t -> if fst h = s then Some(snd h)
             else look t s
let empty () : 'a alist = []
```

Bemærk:

- Vi kan benytte den tomme liste `[]` til at repræsentere den tomme associationsliste.
- Vi skal senere se hvorledes vi med moduler kan sikre at typen `'a alist` bliver "fuldt abstrakt" således at kun de nævnte funktioner kan benyttes til at opererere på de konstruerede associationslister.

Record-typer (generaliserede produkter)

Records i F# giver mulighed for at navngive elementer i et tuple.

Syntaksen for at definere en record-type er enkel:

```
type person = {first:string; last:string; age:int}

let xs = [{first="Lene"; last="Andersen"; age=56};
          {last="Hansen"; first="Jens"; age=39}]
let name (p:person) : string = p.first + " " + p.last
let incr_age (p:person) : person = {p with age=p.age+1}
let ys = List.map incr_age xs
```

Bemærk:

- Ved konstruktion af en record er felt-rækkefølgen ubetydelig.
- Elementer i en record kan udtrækkes ved brug af **dot-notationen** (`p.first`).
- En **ny** record kan konstrueres (med et opdateret element) ved brug af `with`-konstruktionen.

Konklusion

- Typer
- Typeforkortelser
- Record typer (generaliserede produkter)