Programmering og Problemløsning

Typer og Mønstergenkendelse (Del 2)

Martin Elsman

Datalogisk Institut, Københavns Universitet (DIKU)

- Mønstergenkendelse
- Mønstergenkendelse på tupler og records
- Mønstergenkendelse på grundtyper
- Mønstergenkendelse på option-typer
- Mønstergenkendelse på lister

Mønstergenkendelse (Pattern matching)

Generelt set giver mønstergenkendelse mulighed for at **undersøge** og **nedbryde** en værdi i dens bestanddele.

Vi vil se på mønstergenkendelse ud fra typen på de værdier vi undersøger.

I F# kan mønstergenkendelse optræde i flere forskellige programkonstruktioner:

- I simple `let`-bindinger.
- I `match-with`-konstruktioner.
- I funktionsparametre.

Mønstergenkendelse på Tupler

```
let x = (34,"hej",2.3)    // construct triple
let (_,b,f) = x           // use of wildcard (_)
do printfn "%s:%f" b f    // b and f are available here
```

Mønstergenkendelse på Records

```
type person = {first:string; last:string; age:int}
let name ({first=f;last=l}:person) = f + " " + l
```

Bemærk:

1. Matching på records kræver blot at et udvalg af felt-navne er nævnt.
2. Hvis flere record-typer benytter samme felt-navne kan det være nødvendigt med type-annoteringer.
3. Mønstergenkendelse for tupler og records er også meget anvendte i funktionsparametre:

```
let swap (x:'a,y:'b) : 'b * 'a = (y,x)
```

Mønstergenkendelse på heltal og andre grundtyper

Implementation af Fibonacci med mønstergenkendelse:

```
let rec fib n =
  match n with
  | 1 -> 1
  | 2 -> 1
  | _ -> fib(n-1) + fib(n-2)
let v = fib 10
```

Bemærk:

1. Den første bar (|) i en match-case er optional.
2. Den første match-case der "matcher" vinder.
3. Wildcards (_) kan benyttes i en match-case.
4. Tilsvarende kan der matches på andre grundtyper, såsom karakterer, booleans og strenge.

Mønstergenkendelse på option-værdier

Typen `int option` er et eksempel på en simpel såkaldt "sum-type", også kaldt "discriminated union", som repræsenterer værdier der enten er værdien None eller er en værdi Some(*v*), hvor *v* er en værdi af typen `int`.

Her er en funktion der "løfter" addition til værdier af typen `int option`:

```
let add_opt (a:int option) (b:int option) : int option  =
  match a, b with
  | Some a, Some b -> Some(a+b)
  | _ -> None
```

Bemærk:

1. Der benyttes her en form for "nested pattern matching" på par af værdier af typen `int option`.
2. Ved konstruktion og matching af tupler kan man ofte undvære brugen af parenteser.
3. Variabler kan **bindes** i en match-case og henvises til i højre-siden af en match, hvor de vil varetage de matchede værdier.

Mønstergenkendelse på lister

Liste-værdier **konstrueres** grundlæggende set ved brug af to forskellige konstruktører:

`[]`	(Nil)	Konstruktion af den tomme liste.
`x::xs`	(Cons)	Konstruktion af et listeelement med hovedet `x` og halen `xs` (en anden liste).

Samme to konstruktører benyttes ved mønstergenkendelse på en liste:

```
let rec length (l: 'a list) : int =
  match l with
  | [] -> 0
  | x::xs -> 1 + length xs
```

Mønstergenkendelse med `function`-konstruktionen

```
let rec length : 'a list -> int =
  function [] -> 0
         | x::xs -> 1 + length xs
```

Bemærk: Funktionsparameter og **`match`-`with`**-konstruktionen sammentrækkes.

Nestede mønstergenkendelser på lister

Liste-værdier kan matches til dybere niveauer end første cons-celle:

```
let lengthy (l: 'a list) : bool =
  match l with
  | _::_::_ -> true  // at least two cells
  | _ -> false
```

Mønstre kan være mere komplekse:

```
let rec ones (l: int list) : int =
  match l with
  | [] -> 0
  | 1::xs -> 1 + ones xs  // match-cases are tested in order
  | _::xs -> ones xs
```

Konklusion

- Mønstergenkendelse
- Mønstergenkendelse på tupler og records
- Mønstergenkendelse på grundtyper
- Mønstergenkendelse på option-typer
- Mønstergenkendelse på lister