Mere om rekursive typer

Torben Ægidius Mogensen

PoP 14102016

Lister som rekursive typer

Listetypen `'a list` er også en rekursiv type:

```
type 'a list = [] | 'a :: 'a list
```

Ovenstående kan ikke skrives i F#, da konstruktornavne skal starte med stort bogstav, og infix konstruktorer er ikke tilladt. Men man kan uden videre skrive en ækvivalent definition:

```
type 'a liste = Nil | Cons of 'a * 'a liste
```

hvor `Nil` svarer til `[]`, og `Cons` svarer til en præfixudgave af `::`.

Funktioner på vores egen listetype

Vi kan sammenligne funktionsdefinitioner på `'a list` med funktionsdefinitioner på `'a liste`:

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

let rec listeLength (xs : 'a liste) =
  match xs with
  | Nil -> 0
  | Cons (x, xs) -> 1 + listeLength xs

let rec listMap f (xs : 'a list) =
  match xs with
  | [] -> []
  | x :: xs -> (f x) :: listMap f xs

let rec listeMap f (xs : 'a liste) =
  match xs with
  | Nil -> Nil
  | Cons (x, xs) -> Cons (f x, listeMap f xs)
```

Sammenligning af opremsede typer

Hvis vi har typen:

```
type day = Mon | Tue | Wed | Thu | Fri | Sat | Sun
```

kan man sammenligne værdierne med =, <, osv:

```
Mon = Mon    ⇝  true
Mon = Tue    ⇝  false
Mon < Mon    ⇝  false
Mon < Tue    ⇝  true
Mon <= Fri   ⇝  true
Mon <> Fri   ⇝  true
```

Reglen er: Hvis et navn kommer før et andet i opremsningen, er det en mindre værdi.

Sammenligning i option-typen

Vi erindrer:

```
type 'a option = None | Some of 'a
```

Værdier af typen *a* `option` kan sammenlignes, hvis værdier af typen *a* kan sammenlignes:

`None = None`	$\rightsquigarrow$	`true`
`None < Some` x	$\rightsquigarrow$	`true`
`None > Some` x	$\rightsquigarrow$	`false`
`Some` x `= Some` y	$\Leftrightarrow$	$x = y$
`Some` x `< Some` y	$\Leftrightarrow$	$x < y$

Reglen er: `None` kommer før `Some` x, og to værdier med `Some` sammenlignes ved at sammenligne argumenterne til `Some`.

Rækkefølgen har betydning!

Hvis vi laver en alternativ option-type:

```
type 'a opt = Som of 'a | Non
```

gælder:

`Non = Non`	$\rightsquigarrow$	`true`
`Non < Som` x	$\rightsquigarrow$	`false`
`Non > Som` x	$\rightsquigarrow$	`true`
`Som` x `= Som` y	$\Leftrightarrow$	$x = y$
`Som` x `< Som` y	$\Leftrightarrow$	$x < y$

De generelle regler for sammenligning af opremsede typer / sumtyper er:

- Hvis konstruktorerne er forskellige, sammenlignes de efter rækkefølgen i erklæringen.
- Hvis konstruktorerne er ens, sammenlignes de ved at sammenligne argumenterne.

Eksempler på brug af ordning

Man kan bruge sammenligning af værdier i opremsede typer til at skrive korte definitioner af nogle funktioner:

```
type day = Mon | Tue | Wed | Thu | Fri | Sat | Sun

let dayToNumber d =
  List.length
    (List.filter ((>=) d) [Mon; Tue; Wed; Thu; Fri; Sat; Sun])

let nextDay d =
  match
    List.tryFind ((<) d) [Mon; Tue; Wed; Thu; Fri; Sat; Sun]
      with
  | None -> Mon
  | Some d' -> d'
```

Ovenstående er dog væsentligt mindre læselige end de oplagte definitioner, så de er ikke oplagt bedre.

Eksempler med figurtypen

```
type point = int * int // (x, y)
type colour = int * int * int  // (red, green, blue)

type figure =
        | Circle of point * int * colour
          // center, radius, colour
        | Rectangle of point * point * colour
          // bottom-left, top-right, colour
        | Mix of figure * figure

// rotate figure 90° around (0,0)
let rec rotate90 f =
  match f with
  | Circle ((x, y), r, c) -> Circle ((-y, x), r, c)
  | Rectangle ((x1, y1), (x2,y2), c) ->
        Rectangle ((-y2, x1), (-y1,x2), c)
  | Mix (f1, f2) -> Mix (rotate90 f1, rotate90 f2)
```

Funktion, der laver figur

Laver spiral af cirkler.

```
let rec makeSpiral (cx, cy) (x, y) radius colour =
  let circle = Circle ((x + cx,y + cy), int radius, colour)
  if radius < 1.2 then circle
  else
    let (ss, cc) = (0.99 * sin 0.2, 0.99 * cos 0.2)
    let newX = int (cc * float x - ss * float y)
    let newY = int (ss * float x + cc * float y)
    let newRadius = radius * 0.95
    let (r, g, b) = colour
    let newColour = (r+(255-g)/10, g+(255-b)/10, b+(255-r)/10)
    Mix (circle,
         makeSpiral (cx, cy) (newX, newY) newRadius newColour)

makePicture "fractal"
            (makeSpiral (204, 180) (120,120) 100.0 (200,0,0))
            440 440
```

Resultatet