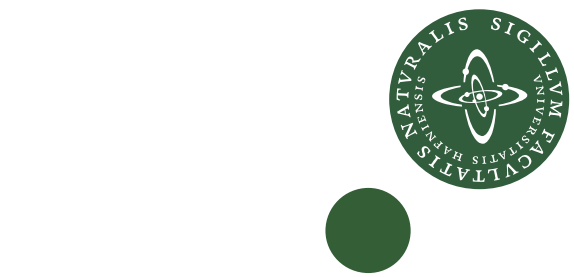
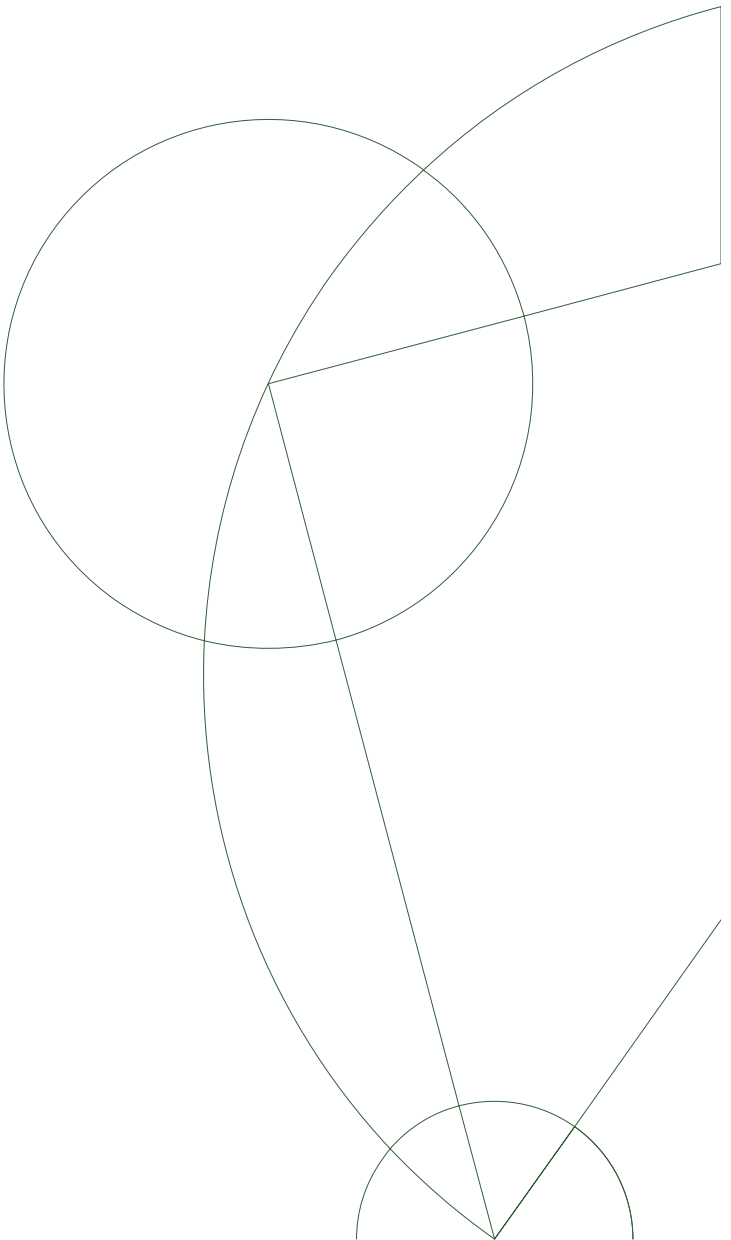


Programmering og Problemløsning

Aflevering 6g



Adam Ingwersen,
Aske Fjellerup
Peter Friborg

Datalogisk Institut
Københavns Universitet

November 6, 2016

1 Delopgave: g6.1

Denne opgave stifter vi bekendtskab med typer i F#. Opgaven løses ved at lave en type `weekday`, med 7 parametre: `Monday... Sunday`.
Vi laver to funktioner:

- **dayToNumber n**: der givet en ugedag giver ugedagensnummer.
  - forventning `dayToNumber Monday` returnerer `1`.
- **numberToDay n**: der givet et nummer 0<n<8 giver ugedagen, som en `option type`.
  - forventning `numberToDay 4` returnerer `Some Thursday`.

Vores to funktioner er lavet med pattern matching.
Vi forventer at vores funktion med inputtet: `numberToDay (dayToNumber Wednesday)` returnerer `Some Wednesday`.

2 Delopgave: g6.2

Vi definerede i øvelsesopgaverne en figur o61, der var af typen `Mix of Circle * Rectangle`.
Vi laver endnu en figur, af typen `Twice of figure * Point`.
Hvilket netop er en figur af typen `Figure` forskudt med en vektor, `(x,y)`.

```
let g61 =
    Twice (o61, (50, 70))
```

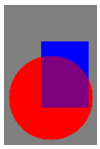


Figure 1: Figuren fra øvelsesopgave Ø6.5

3 Delopgave: g6.3

I funktionen `colourAt` tilføjes et argument yderligere til vores pattern matching. Vores umiddelbare tanker ifht. at løse problemet er, at konstruere Twice i colourAt, der gør:

1. Tag inputs i form af en figur, m, og et sæt af koordinater, (int1, int2)
2. Hvis ingen af figurerne overlapper - vis farven som normalt
3. Hvis figurerne overlapper: Så tegn farven for den seneste figur, altså den forskudte.

Implementeringen af ovenstående design, sker ved; at inkoorporere definitionen af Twice til pattern-match i colourAt. Første kriterium, input, løses ved at lave endnu et pattern-match. Matchet matcher en tuple, af den oprindelige figur og den forskudte figur. Den forskudte figurs koordinater findes ved at trække vores vektor `(x1, y1)` fra de oprindelige koordinater sådan at `((x - x1), (y - y1))`. De enkelte matches ligner matchene fra `colourAt Mix`, men istedet for at tage gennemsnittet af farverne, når figurerne overlapper, tager vi kun farven fra den forskudte figur `(Some (r1,g1,b1), Some (r2,g2,b2)) -> Some (r2,g2,b2)`

```
| Twice (figure, (x1, y1)) ->
    match (colourAt (x, y) figure, colourAt ((x - x1)
        , (y - y1)) figure) with
    | (None, c) -> c
    | (c, None) -> c
    | (Some (r1,g1,b1), Some (r2,g2,b2)) -> Some (r2,
      g2,b2)
```

Dette giver det visuelle resultat, at den forskudte figur ligger øverst.

4 Delopgave: g6.4

Vi anvender makeBMP.dll-filen sammen med termninal-kaldet `fsharpi -r makeBMP.dll 6g4.fsx` og konverterer til .png vhja. `convert g64.bmp g64.png`. Nedenstående er resultatet - som ønsket.

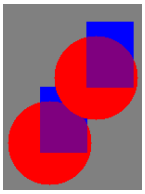


Figure 2: g64.bmp → g64.png

5 Delopgave: g6.5

checkFigure

Her skal vi arbejde videre med en funktion fra øvelsesopgave ø6.6. Der givet en `figure` vil returnere, om det er en lovlig figur mht. farve og længder.
`checkFigure` er en funktion der bruger pattern matching til at matche med den rigtige type:

- | **Circle**: tjekker om radius er positiv og at farverne ligger mellem 0 og 255.
- | **Rectangle**: tjekker, om det nederste venstre hjørne vitterligt er det nederste venstre hjørne og at farverne ligger mellem 0 og 255.
- | **Mix (f1, f2)**: er to rekursive kald, med hhv. (checkFigure f1) og (checkFigure f2).
- | **Twice (f1, point)**: er et enkelt rekursivt kald, med den ene figure som argument.

```
  | Twice (f3, (_, _)) -> (checkFigure f3)
```

For at sikre hensigtsmæssig kode, har vi opstillet et system for black-box testing af funktionen checkFigure. Vi har opsat nogle ugyldige såvel som gyldige figurer og forventer at:

```
let bbf1 =
  let cirkel = Circle ((50,50), -10, (255, 0, 0))
  let firkant = Rectangle ((40, 40), (90, 110), (0, 0,
     250))
  Mix (cirkel, firkant)

let bbf2 =
  let cirkel = Circle ((50,50), 45, (255, 0, 0))
  let firkant = Rectangle ((40, 40), (90, 110), (0,
     500, 250))
```

...giver false i begge tilfælde.

```
let bbt1 =
  let cirkel = Circle ((50,50), 45, (255, 0, 0))
  let firkant = Rectangle ((40, 40), (90, 110), (0, 0,
     250))
  Mix (cirkel, firkant)

let bbt2 =
  Twice (bbt1, (50, 70))
```

... giver true i begge tilfælde.
Ved kørsel af 6g-2345.fsx, ses det, at funktionen fungerer som forventet.

boundingBox

Her laver vi en udvidelse til funktionen boundingBox fra opgave ø6.8 så den også fungerer til **figure : Twice**.
For at finde den mindste rektangel i en figur at typen `Twice` findes den midste rektangel i den nye klon af figuren, da den gamle har en god chance for at ikke at være en rektangel mere og ellers ville bare have samme størelse.

```
|Twice (f1,(a,b)) -> match (boundBox f1) with
                      |((x1,y1),(x2,y2)) -> ((x1+a,y1+b)
                          ,(x2+a,y2+b))
```

Vi har opsat nogle figurer og forventer at:

```
boundBox(o61) -> ((40, 40), (90, 110))

boundBox(g61) -> ((90, 110), (140, 180))
```

Ved blackbox-testing ses det, at funktionen virker som forventet.