Introduktion til Programmering og Problemløsning (PoP)

Videregående emner

Jon Sporring
Department of Computer Science
2020/11/25

UNIVERSITY OF COPENHAGEN

Rettelse til arbejdsseddel 9

"9g2 First extend the library readNWrite.fs with a function,

tac : filenames:string list -> string option

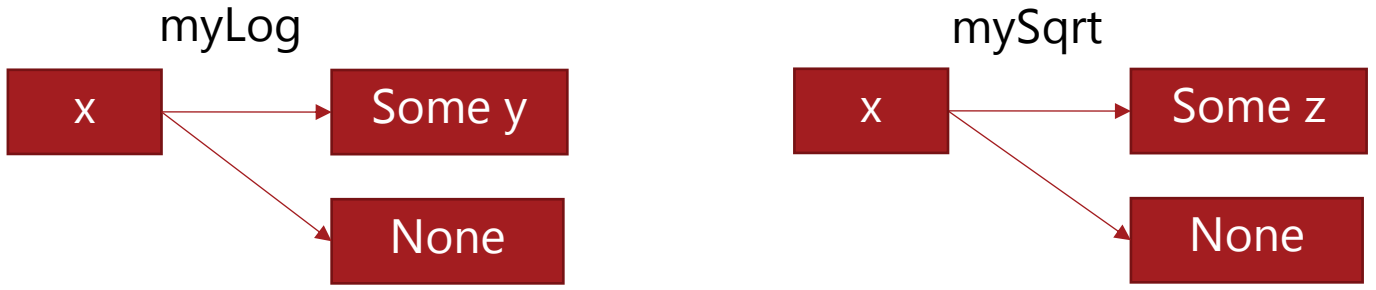
which takes a list of files, reads their content with readFile (Exercise 9.1g0), reverses the order of each file in a line-by-line manner and reverses each line (i.e. the opposite of cat) and concatenates the result. If any of the files do not exist, then the function should return None."

Altså: vend rækkefølgen af filer, vend rækkefølgen af linjer, vend hver linje. Udskriv linje for linje.

Sammensætning med option typer

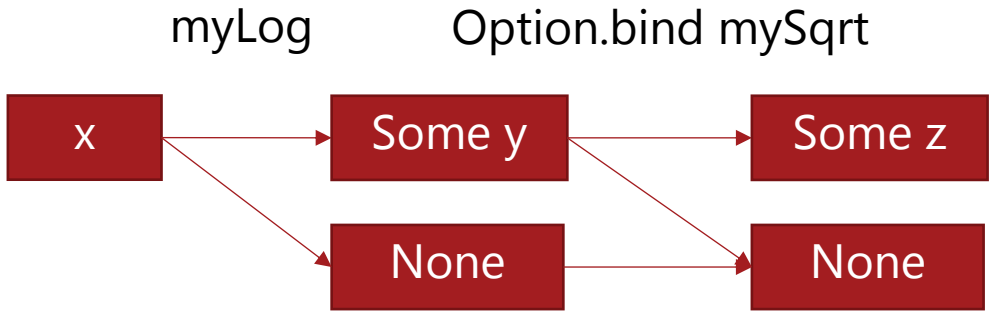
- Følgende funktioner håndterer evt. fejl med option-typer:

```
let myLog x : float option = if x > 0.0 then Some (log x) else None
let mySqrt x : float option = if x >= 0.0 then Some (sqrt x) else None;;
```



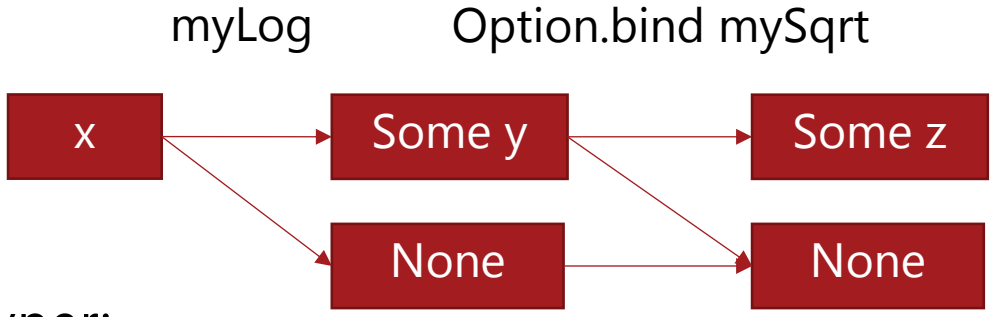
- Denne funktionalitet er allerede tilgængelig med `Option.bind` funktionen:

```
val bind : (('a -> 'b option) -> 'a option -> 'b option)
```




```
1.0 |> myLog |> Option.bind mySqrt
val it : float option = Some 0.0
```

Sammensætning med option typer



- Følgende funktioner håndterer evt. fejl med option-typer:

```
let myLog x : float option = if x > 0.0 then Some (log x) else None
let mySqrt x : float option = if x >= 0.0 then Some (sqrt x) else None;;
```

- Med `Option.bind` havde vi

```
1.0 |> myLog |> Option.bind mySqrt
```

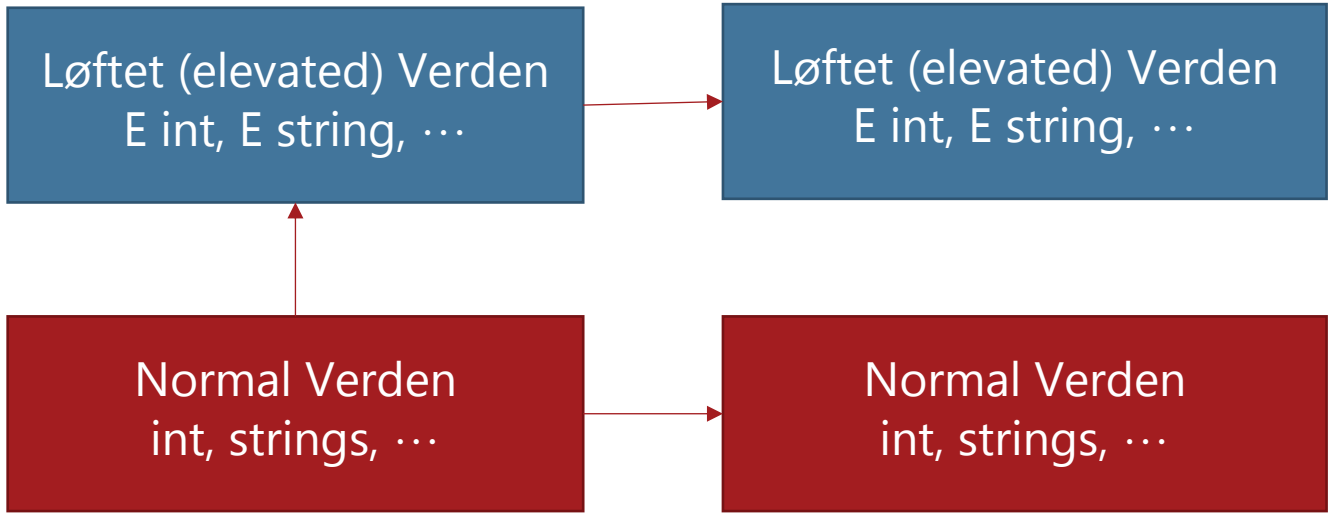
- Operator symboler: `!, %, &, *, +, -, ., /, <, =, >, ?, @, ^, |, ~.`
  Infix med mindre foranstillet med `!` (pånær `!=`) eller en eller flere `~`, så `prefix`.

- Infix operator `>>=`:

```
let (>>=) x f = Option.bind f x
1.0 |> myLog >>= mySqrt
val it : float option = Some 0.0
```

Infix: a + b
Prefix: -a

Monadisk struktur til funktionskomposition



Abstraheret program

```
1.0 |> myLog >>= mySqrt
```

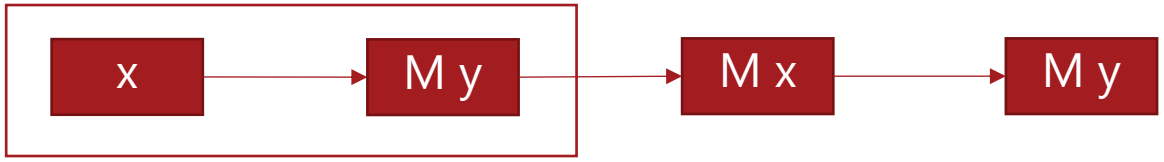
Kompliceret programlogik

```
let mySqrtLog x : float option =
  let logX = myLog x
  match logX with
    None -> None
    | Some y -> mySqrt y
mySqrtLog 1.0;;
```

Monadisk værdi

Monadisk funktion

Monadisk bind



Sammensætning med List

- Følgende funktioner laver lister (normal til løftet verden)

```
let iter x = [1..x]
let dbl x = [x; x]
```

- `List.collect` er en `bind`-funktion:

  val List.collect : (('a -> 'b list) -> 'a list -> 'b list)

```
let (>>=) x f = List.collect f x
3 |> iter >>= dbl;;
val it : int list = [1; 1; 2; 2; 3; 3]
```

- `powerset`: Sættet af alle delsæt

```
let (>>=) x f = List.collect f x
let rec powerset lst =
  match lst with
    [] -> [[]]
    | x::xs -> xs |> powerset >>= (fun ys -> [ys; x::ys])
powerset [1;2;3];;
val it : int list list =
  [[]; [1]; [2]; [1; 2]; [3]; [1; 3]; [2; 3]; [1; 2; 3]]
```

Spørgsmål

- Kan man lave et module bestående af flere .fs filer?
  - Nej ikke nemt, men man kan udvide funktionaliteten

mod.fs

```
module myMod
let f (x:float) : float = x*x
```

app.fsx

```
module myMod =
  let g (x:float) : float = sqrt x

printfn "%A" (myMod.f 3.0)
printfn "%A" (myMod.g 3.0);
```

- Hvornår giver det mening at bruge expressions som i opgave 7ø11-15?
  - Besvaret i gennemgang tidligere i dag.

Spørgsmål

- Hvordan laver man mere avanceret pattern matching (f.eks på to værdier på en gang, mere komplicerede typer som (a' * b') option, samt brug af guards)?
  - Lyv om Jons alder:

```
let getAge person =
    match person with
        (age, name) when name = "Jon" -> 24
        | (age, name) -> age
printfn "%A" (getAge (46,"Martin"))
printfn "%A" (getAge (51,"Jon"))
```

- Kan vi pattern matche på sidste element i en liste ?
  - Nej. Hvor ville det være en dårlig ide?
- Spørgsmål om wargame
  - Uddelegeret til Martin.