Introduktion til Programmering og Problemløsning (PoP)

Undtagelser og Option-typer

Jon Sporring
Department of Computer Science
2020/11/22

UNIVERSITY OF COPENHAGEN

Håndtering af fejltilstande

Undtagelser (exceptions):
- Undtagelsesværdier
- Kaste en undtagelse (raise/throw)
- Håndtering af undtagelser (try-with)
- Indbyggede exceptions og hjælpefunktioner

Alternativer til undtagelser:
- Fejlhåndtering med option typer.

```
let str = "hej"
for i = 1 to 3 do
  printf "%c " str.[i]
printfn "";;

e j System.IndexOutOfRangeException: Index was outside the bounds of the array.
  at <StartupCode$FSI_0007>.$FSI_0007.main@ () [0x0002a] in
<a92c879f677b49e985c4e89c0484d9c8>:0
  at (wrapper managed-to-native)
System.Reflection.RuntimeMethodInfo.InternalInvoke(System.Reflection.RuntimeMethodInfo,obje
ct,object[],System.Exception&)
  at System.Reflection.RuntimeMethodInfo.Invoke (System.Object obj,
System.Reflection.BindingFlags invokeAttr, System.Reflection.Binder binder, System.Object[]
parameters, System.Globalization.CultureInfo culture) [0x0006a] in
<55adae4546cd485ba70e2948332ebe8c>:0
Stopped due to error
```

Undtagelsesværdier

- Undtagelser er værdier af den indbyggede *udvidbare* type `exn`.
- Undtagelsesværdier kan være konstante værdier eller bære argumenter.
- Nye undtagelseskonstruktører kan erklæres med `exception`-konstruktionen:

```
exception MyError
exception MyArgExn of int
let e1 : exn = MyError
let e2 : exn = MyArgExn 5
```

- Undtagelsesværdier tillader lighed:

```
let isMyError = e1 = e2 // false
```

- Undtagelsesværdier kan benyttes i matches:

```
match e2 with
  MyArgExn 5 -> "yes”
  | _ -> "no”              // ”yes”
```

Kaste en undtagelse

- Undtagelser afbryder det normale kontrol-flow.
- Konstruktionen der benyttes til at "*kaste eller rejse en undtagelse*":

```
val raise : System.Exception -> 'a
```

- **Bemærk:** `raise` returner en vilkårlig værdi, men den bliver aldrig brugt, da funktionen aldrig returnerer, men derimod sender en besked (en undtagelse) "op i kaldstakken" om at beregningen blev afbrudt.

```
let myfun a =
  if a then 2 else raise (MyExnArg 5)
myfun false;;
```

```
FSI_0027+MyExnArg: Exception of type 'FSI_0027+MyExnArg' was thrown.
  at FSI_0029.myfun (System.Boolean a) [0x00019] in
<a92c879f677b49e985c4e89c0484d9c8>:0
  at <StartupCode$FSI_0029>.$FSI_0029.main@ () [0x00000] in
<a92c879f677b49e985c4e89c0484d9c8>:0
  at (wrapper managed-to-native)
System.Reflection.RuntimeMethodInfo.InternalInvoke(System.Reflection.RuntimeMethodInfo
,object,object[],System.Exception&)
  at System.Reflection.RuntimeMethodInfo.Invoke (System.Object obj,
System.Reflection.BindingFlags invokeAttr, System.Reflection.Binder binder,
System.Object[] parameters, System.Globalization.CultureInfo culture) [0x0006a] in
<55adae4546cd485ba70e2948332ebe8c>:0
Stopped due to error
```

Undtagelser kan fanges

- Det er muligt at *fange* kastede undtagelser på et højere niveau ved at benytte `try-with`:

```
let myfun a =
  if a then 2 else raise (MyExnArg 5)
let y =
  try myfun false with
    MyExnArg x -> x
    | _ -> 0;;
val y : int = 5
```

- Typen i try-delen skal være den samme som i `with`-grenene.

Indbyggede Exceptions og Hjælpefunktioner

- For at matche de indbyggede Mono exceptions, kan det være nødvendigt at benytte *dynamic type matching*, som benytter sig af følgende syntax:

```
let mydiv a b : int =
  try a / b with
    :? System.DivideByZeroException -> 0
mydiv 12 0;;

val it : int = 0
```

- Nogle hjælpefunktioner:

```
val failwith : string -> 'a
val invalidArg : string -> string -> 'a
```

```
let toFahrenheit c =
  if c < -273.15 then invalidArg "c" "below absolute zero"
  else 9.0/5.0*float(c)+32.0
```

Fejlhåndtering med option typer

- Option-typer kan bruges til at indkode exceptionel opførsel. Funktionen mydiv returnerede 0 ved fejl, hvor resultatet burde være "udefineret" eller None:

```
let mydiv a b : int option =
  try Some (a / b) with
    :? System.DivideByZeroException -> None
mydiv 12 0;;

val mydiv : a:int -> b:int -> int option
val it : int option = None

mydiv 12 4;;
val it : int option = Some 3
```

- Option-typer match'es:

```
let v = Some 3
match v with
  Some x -> x
  | None -> 0   // 3
```

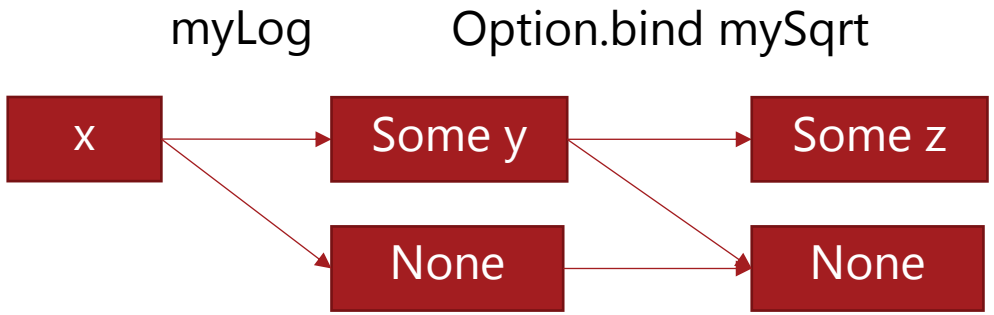
Sammensætning med option typer

- 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;;
```

- Deres direkte sammensætning er besværlig:

```
let mySqrtLog x : float option =
  let logX = myLog x
  match logX with
    None -> None
    | Some y -> mySqrt y
mySqrtLog 1.0;;
val it : float option = Some 0.0
```



- 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
```

Resumé

I denne video hørte du om:

- Undtagelser (exceptions):
  - Undtagelsesværdier
  - Kaste en undtagelse (raise/throw)
  - Håndtering af undtagelser (try-with)
  - Indbyggede exceptions og hjælpefunktioner

- Alternativer til undtagelser:
  - Fejlhåndtering med option typer.