Programmering og Problemløsning, 2019

Exceptions (Undtagelser)

Martin Elsman

Datalogisk Institut
Københavns Universitet
DIKU

26. November, 2019

Exceptions

Emner for i dag:

1. **Exceptions (undtagelser):**
   - Exception-værdier.
   - Rejsning af exceptions (`raise`/throw)
   - Håndtering af exceptions (**`try-with`**)
   - Indbyggede exceptions.
   - Nogle nyttige hjælpefunktioner.
2. **Alternativer til Exceptions.**
   - Fejlhåndtering med `option` typer.
   - Beskedbærende fejltyper.

Exception-værdier

Exceptions er værdier af den indbyggede *udvidbare* type `exn`.

Exception-værdier kan være konstante værdier, der ikke bærer argumenter, og exception-værdier der bærer argumenter.

Nye *exception-konstruktører* kan erklæres med **`exception`**-konstruktionen:

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

Exception-værdier tillader lighed:

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

... og kan benyttes i matches:

```
let isMyArgExn =
  match e2 with MyArgExn _ -> "yes"        // ⇝ "yes"
              | _ -> "no"
```

Rejsning af Exceptions

Exceptions kan benyttes til at afbryde det normale kontrol-flow (deraf navnet exception).

Konstruktionen der benyttes til at "*rejse en exception*" er operationen `raise`:

```
val raise : exn -> 'a
```

Bemærk at operationen kan instantieres til at returnere en værdi af vilkårlig type, hvilket skyldes at operationen aldrig returnerer, men derimod sender en besked (en exception) "op i kaldstakken" om at beregningen blev afbrudt.

Det er muligt at *håndtere* rejste exceptions på et højere niveau ved at benytte en **`try-with`** konstruktion.

Her er hvad der sker hvis en rejst exception ikke håndteres:

```
- exception MyArgExn of int;;
- "hello " + raise (MyArgExn 42)
FSI_0002+MyArgExn: Exception ... was thrown...
Stopped due to error
```

Den rejste exception blev "håndteret" af `fsharpi`'s *top-level handler*.

Håndtering af rejste exceptions

Rejste exceptions kan håndteres på et højere niveau i programmet ved at benytte en **try-with** konstruktion.

Eksempel (exn.fs):

```
exception MyExnArg of int
let f () = if false then 8 else raise (MyExnArg 5)
let y = try f () with MyExnArg x -> x
do printfn "%d" y
```

Kørsel:

```
bash-3.2$ fsharpc --nologo exn.fs && mono exn.exe
5
```

Bemærk:

Argumentet til den rejste exception blev trukket ud af exception-værdien ved brug af *exception pattern matching*.

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 option =
  try Some (a / b) with
    :? System.DivideByZeroException -> None
```

Nogle hjælpefunktioner:

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

Brug af funktionen `invalidArg`:

```
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

Funktionen `mydiv` benytter værdien None til at indikere en fejl.

Option-typer kan således bruges til at indkode exceptionel opførsel.

Funktionen `Option.bind` kan bruges til at styre sammensætningen af beregninger:

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

Eksempel

```
> let (>>=) x y = Option.bind y x
> mydiv 8 3 >>= (fun x -> Some(float(x)+1.0));;
val it : float option = Some 3.0
```

Bemærk:

- Funktionen `Option.bind` er et simpelt eksempel på bindingsoperatoren i en såkaldt *monad*, en abstraktionsmekanisme der giver mulighed for at sammensætte effektfulde beregninger på en sund måde.
- Monads ligger blandt andet til grund for sammensætning af effektfulde beregninger i Haskell.

Beskedbærende fejl-typer

Det er muligt at udvidde teknikken med funktionalitet der bærer information om fejlen.

```
type 'a result = Ok of 'a | Error of string
val (>>=) : 'a result -> ('a -> 'b result) -> 'b result
```

Eksempel

```
let mydiv a b : int result =
  try Ok (a / b) with
    :? System.DivideByZeroException -> Error "div"

let (>>=) a f =
  match a with Ok v -> f v
               | Error s -> Error s

do printfn "%A" (mydiv 8 3 >>= (fun x -> Ok(float(x)+1.0)))
```

Bemærk:

- Funktionaliteten kan også udviddes til at understøtte tilfælde hvor det er muligt at opsamle og rapportere multiple fejl.