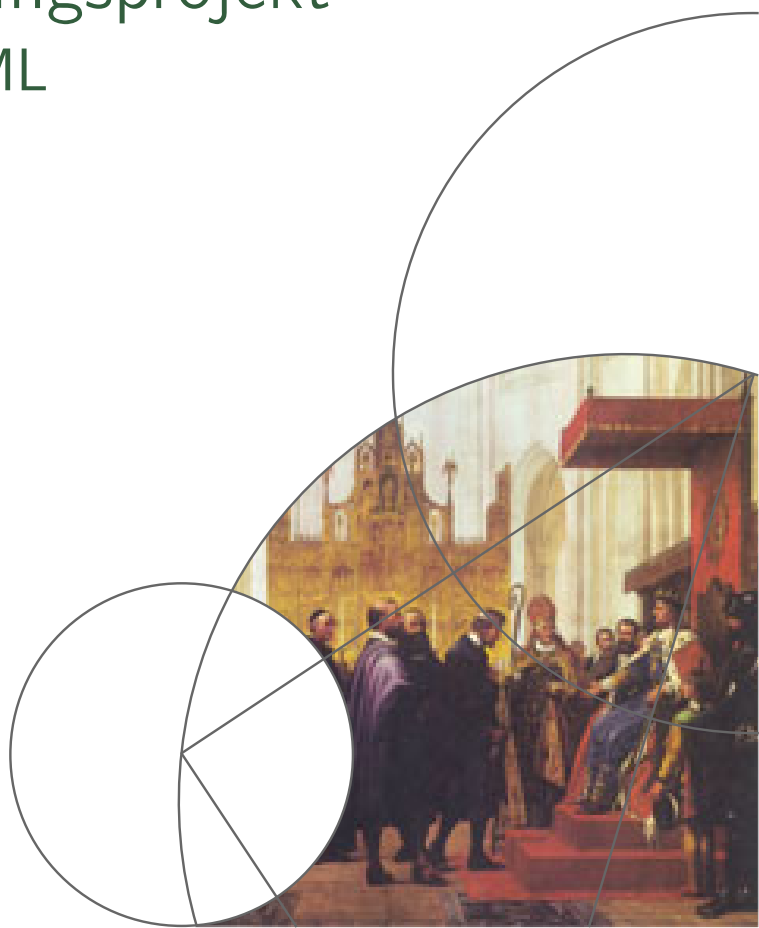
Brug SML til jeres systemudviklingsprojekt

Om at skrive større programmer med SML

Morten Brøns-Pedersen (f@ntast.dk)
Datalogisk institut



Program

- Introduktion (med eksempel).
- Kort om systemdesign.
- SML's modulsystem.
- Et par eksempler.
- Det ~~kedelige~~ praktiske.
- Afrunding.

Diaser og kode kan findes på
`http://github.com/mortenbp`.

Introduktion

På kurset Introduktion på Programmering har I lært SML at kende. Men de problemer I har løst har været små og veldefinerede.

Målet for i dag er at gøre jer i stand til

> At strukturere, og udvikle, et større program med SML.

Vi starter med et eksempel.

Eksempel: Persondatabase (1/13)

Designovervejelser

- Hvad skal den kunne? Vi kan f.eks. lave hurtigt opslag af personer på navn eller CPR-nummer, men ikke umiddelbart begge.
- Hvor generisk skal databasen være? Er det bare et tilfælde af en mængde eller en afbildning, eller skal den være mere specialiseret?
- Hvad er formålet med databasen? Formålet skal afspejles i designet; *godt design kendes på hvad der er udeladt.*
- Skal det være muligt at arbejde med flere databaser, eller findes der kun én enkelt?

Eksempel: Persondatabase (2/13)

MyFaceTube©

En database som holder styr på venskaber mellem personer. Venskaber er ikke nødvendigvis gensidige.

Der findes kun én **MyFaceTube©**!

MyFaceTube© er *ikke*

- En telefonbog.
- Et stamtræ.
- Et fotoalbum.
- ...

Ekstra funktionalitet kan implementeres som en ny database eller direkte i **MyFaceTube©**. Endnu et valg at træffe!

Eksempel: Persondatabase (3/13)

Hvilke beregninger og hvilke objekter består **MyFaceTube©** af?

- Personer.
- Mængder (af personer).
- Opslag (fra personer til mængder af personer) — Hvem er dine venner?
- Selve **MyFaceTube©**.

Vi starter med signaturerne.

Eksempel: Persondatabase (4/13)

```
1  signature ORDNING =
2  sig
3    type t
4
5    val sammenlign : t * t -> order
6  end
```

Eksempel: Persondatabase (4/13)

```
1  signature ORDNING =
2  sig
3    type t
4
5    val sammenlign : t * t -> order
6  end
```

Husk at `order` er defineret i standardbiblioteket:

```
datatype order = LESS | EQUAL | GREATER
```

Eksempel: Persondatabase (5/13)

```
1  signature PERSON =
2  sig
3    type t
4
5    val ny : {navn : string, cpr : int} -> t
6    val navn : t -> string
7    val cpr : t -> int
8
9    val sammenlign : t * t -> order
10 end
```

Eksempel: Persondatabase (6/13)

```
1  signature ORDNET_MAENGDE =
2  sig
3    type element
4    type t
5
6    val tom : t
7    val indsaet : t -> element -> t
8    val indeholder : t -> element -> bool
9    val faelles : t -> t -> t
10   val fjern : t -> element -> t
11   val erTom : t -> bool
12
13   (* Refleksiv variant: {delmaengde a b} betyder a \subseteq b *)
14   val delmaengde : t -> t -> bool
15
16   (* Kan hejse {Empty} *)
17   val tagMindste : t -> element * t
18   val tagStoerste : t -> element * t
19 end
```

Eksempel: Persondatabase (7/13)

```
1  signature ORDNET_OPSLAG =
2  sig
3    type noegle
4    type 'a t
5
6    val tom : 'a t
7    val indsaet : 'a t -> noegle * 'a -> 'a t
8    val find : 'a t -> noegle -> 'a option
9    val fjern : 'a t -> noegle -> 'a t
10
11   (* Kan hejse {Empty} *)
12   val tagMindste : 'a t -> (noegle * 'a) * 'a t
13   val tagStoerste : 'a t -> (noegle * 'a) * 'a t
14 end
```

Eksempel: Persondatabase (8/13)

```
1  signature MY_FACE_TUBE =
2  sig
3    val tilmeld : Person.t -> unit
4    val afmeld : Person.t -> unit
5    val medlem : Person.t -> bool
6    val medlemmer : unit -> PersonMaengde.t
7
8    (* Hvis ikke medlem returneres den tomme maengde *)
9    val venner : Person.t -> PersonMaengde.t
10
11   (* {nyVen a b} tilfoejer b til a's venner *)
12   val nyVen : Person.t -> Person.t -> unit
13 end
```

Eksempel: Persondatabase (9/13)

Vi gennemgår ikke hele implementeringen her. Koden kan findes online på `http://github.com/mortenbp`.

```
1  structure Person :> PERSON =
2  struct
3  (* Navn * CPR-nummer *)
4  type t = string * int
5
6  (* TODO: Indfoer integritetstjek paa CPR-nummer *)
7  fun ny {navn, cpr} = (navn, cpr)
8
9  fun navn (navn, _) = navn
10 fun cpr (_, cpr) = cpr
11
12 fun sammenlign ((_, x), (_, y)) = Int.compare (x, y)
13 end
```

Eksempel: Persondatabase (10/13)

```
1  functor OrdnetMaengdeFn (Element : ORDNING) :>
2          ORDNET_MAENGDE where type element = Element.t =
3  struct
4  type element = Element.t
5  datatype t = E
6             | T of t * element * t
7
8  val tom = E
9
10 fun indsaet E x = T (E, x, E)
11   | indsaet (T (l, y, r)) x =
12     case Element.sammenlign (x, y) of
13       LESS    => T (indsaet l x, y, r)
14     | EQUAL   => T (l, y, r)
15     | GREATER => T (l, y, indsaet r x)
```

```
structure PersonMaengde = OrdnetMaengdeFn (Person)
```

```
structure PersonOpslag = OrdnetOpslagFn (Person)
```

Eksempel: Persondatabase (11/13)

One of a kind? — Imperativ programmering med SML.

```
1  structure MyFaceTube :> MY_FACE_TUBE =
2  struct
3  structure M = PersonMaengde
4  structure O = PersonOpslag
5
6  val opslag = ref O.tom
7
8  fun tilmeld p =
9      opslag := O.indsaet (!opslag) (p, M.tom)
10
11 fun afmeld p =
12     opslag := O.fjern (!opslag) p
13
14 fun medlem p =
15     case O.find (!opslag) p of
16       SOME _ => true
17     | NONE   => false
18
19 fun medlemmer _ =
```

Eksempel: Persondatabase (12/13)

Live demo.

Eksempel: Persondatabase (13/13)

Alternativ løsning: Lad personer holde styr på deres egne venskaber.

Pro **MyFaceTube©** bliver essentielt blot en mængde af personer — vi behøver ikke implementere opslag.

Contra Vores `Person.t` bliver mindre generel — i mange situationer er det uinteressant at vide hvem der er venner med hvem.

Eksempel: Persondatabase (13/13)

Alternativ løsning: Lad personer holde styr på deres egne venskaber.

Pro **MyFaceTube©** bliver essentielt blot en mængde af personer — vi behøver ikke implementere opslag.

Contra Vores `Person.t` bliver mindre generel — i mange situationer er det uinteressant at vide hvem der er venner med hvem.

```
type t = string * int
```

Eksempel: Persondatabase (13/13)

Alternativ løsning: Lad personer holde styr på deres egne venskaber.

Pro **MyFaceTube©** bliver essentielt blot en mængde af personer — vi behøver ikke implementere opslag.

Contra Vores `Person.t` bliver mindre generel — i mange situationer er det uinteressant at vide hvem der er venner med hvem.

```
type t = string * int
```

```
datatype t = P of string * int * t list
```

Kort om systemdesign

Tommelfingerregel:

> Des mindre to dele af systemet har med hinanden at gøre, des mindre bør de vide om hinanden.

- Opdel systemet i beregning og repræsentation. Det er især vigtigt i funktionel programmering.
- Konventioner omkring repræsentation og invarianter er OK inden for strukturer, *ikke* mellem dem.
- Undgå så vidt muligt at gøre (data)typer synlige for det omgivende program (brug `:>`). Lav i stedet en struktur som modellerer objektet.
- Brug de objekter der passer! Altså; ingen lister hvor der burde være mængder.

SML's modulsystem (1/3)

Signaturer som dokumentation. Eksempel fra MyLib:

```
1  signature Layout =
2  sig
3    (* infix ^^ ++ \ & \\ && *)
4    include Pretty
5
6    (* Prints to standard out (with an extra \n). Takes an optional max width. *)
7    val println : int option -> t -> unit
8
9    (* A space if the output fits, new line and indent if it doesn't. *)
10   val softln : t
11
12   (* Nothing if the output fits, new line and indent if it doesn't. *)
13   val softbrk : t
14
15   (* Replaces all spaces with softln. *)
16   val softtxt : string -> t
17
18   (* Like softtxt but preprends two spaces. *)
19   val paragraph : string -> t
20
21   (* Converts a preformatted text into a document. A newline character separates
22    * paragraphs and the following number of spaces determine the next paragraphs
23    * indentation. So it basically does what you would expect. *)
24   val str : string -> t
```

SML's modulsystem (2/3)

Brug en enkelt struktur for hvert objekt der skal repræsenteres.

Objektets type er `Objekt.t`.

OBS: Strider med standardbibliotekets konvention som er `Objekt.objekt`.

SML's modulsystem (3/3)

Udvidelse af strukturer.

- Signaturer kan inkludere andre signaturer med `include`.

```
1  signature List =
2  sig
3    include LIST
4    val sort : ('a -> 'a -> order) -> 'a list -> 'a list
5    val shuffle : 'a list -> 'a list
```

- Strukturer kan inkludere andre strukturer med `open`.

```
1  structure List :> List =
2  struct
3  open List
```

Et par eksempler

- Når `option` ikke er god nok: `Either.t`.
- Modellering af klasser med funktorer: `FlagFn`.
- Gratis funktioner med funktorer: `FoldbarFn`.
- Doven evaluering med memoisering: `Lazy.t`.

Et par eksempler: `Either.t`

```
1  signature Either =
2  sig
3    datatype ('a, 'b) t = Left of 'a | Right of 'b
4    exception Either
5
6    val ofLeft : ('a, 'b) t -> 'a
7    val ofRight : ('a, 'b) t -> 'b
8    val either : ('a -> 'c) -> ('b -> 'c) -> ('a, 'b) t -> 'c
9    val lefts : ('a, 'b) t list -> 'a list
10   val rights : ('a, 'b) t list -> 'b list
11   val partition : ('a, 'b) t list -> 'a list * 'b list
12 end
```

Et par eksempler: `Either.t`

```
1  structure Either :> Either =
2  struct
3  datatype ('a, 'b) t = Left of 'a | Right of 'b
4  exception Either
5
6  fun ofLeft (Left x) = x
7    | ofLeft _ = raise Either
8
9  fun ofRight (Right x) = x
10   | ofRight _ = raise Either
11
12 fun either l r e =
13     case e of
14       Left x  => l x
15     | Right x => r x
16
17 fun lefts es = List.mapPartial (fn Left x => SOME x | _ => NONE) es
18 fun rights es = List.mapPartial (fn Right x => SOME x | _ => NONE) es
19
20 fun partition es = (lefts es, rights es)
21 end
```

```
1  datatype either = datatype Either.t
2  exception Either = Either.Either
```

Et par eksempler: `FlagFn`

FARE! Imperativ programmering.

```
1  signature FLAG =
2  sig
3    val hejs : string -> unit
4    val saenk : string -> unit
5    val hejst : string -> bool
6  end
```

```
1  functor FlagFn () :> FLAG =
2  struct
3  structure M = TekstMaengde
4  val flag = ref M.tom
5
6  fun hejs f =
7      flag := M.indsaet (!flag) f
8
9  fun saenk f =
10     flag := M.fjern (!flag) f
11
12 fun hejst f =
13     M.indeholder (!flag) f
14 end
```

Et par eksempler: `FlagFn`

FARE! Imperativ programmering.

```
1  structure MineFlag = FlagFn ()
2
3  ;MineFlag.hejs "Danebrog";
```

Et par eksempler: `FoldbarFn`

```
1  signature FOLDBAR = sig
2    type 'a t
3    val fold : ('a * 'b -> 'b) -> 'b -> 'a t -> 'b
4  end
```

```
1  functor FoldbarFn (F : FOLDBAR) :> sig
2    type 'a t = 'a F.t
3    val tilListe : 'a t -> 'a list
4    val alle : 'a t -> ('a -> bool) -> bool
5    val nogle : 'a t -> ('a -> bool) -> bool
6    val findes : ''a t -> ''a -> bool
7    val konkat : 'a list t -> 'a list
8    val anvend : 'a t -> ('a -> unit) -> unit
9  end =
10 struct
11 type 'a t = 'a F.t
12 fun tilListe f = F.fold (fn (a, b) => a :: b) nil f
13 fun alle f g = F.fold (fn (a, b) => g a andalso b) true f
14 fun nogle f g = F.fold (fn (a, b) => g a orelse b) false f
15 fun findes f x = F.fold (fn (a, b) => a = x orelse b) false f
16 fun konkat f = F.fold (fn (a, b) => a @ b) nil f
17 fun anvend f g = F.fold (fn (a, _) => g a) () f
18 end
```

Et par eksempler: `FoldbarFn`

Mange ting er `FOLBAR`e:

- Opslag.
- Træer.
- Lister.
- Grafer.
- Filer.
- ...

De seks funktioner er nu „gratis“ for hver slags objekt.

```
1  structure T = FoldbarFn (Trae)
2  val data = T.tilListe mitTrae
```

Et par eksempler: `Lazy.t`

```
1  (* Memoized lazy evaluation *)
2
3  signature Lazy =
4  sig
5    type 'a t
6    type 'a thunk = unit -> 'a
7
8    val lazy : 'a thunk -> 'a t
9    val force : 'a t -> 'a
10   val eager : 'a -> 'a t
11   val delay : 'a t thunk -> 'a t
12 end
```

Et par eksempler: `Lazy.t`

```
1  structure Lazy :> Lazy =
2  struct
3  type 'a thunk = unit -> 'a
4  datatype 'a t' = Thunk     of 'a thunk
5                 | Value     of 'a
6                 | Exception of exn
7  type 'a t = 'a t' ref
8
9  fun lazy f = ref (Thunk f)
10 fun eager v = ref (Value v)
11 fun force t =
12     case !t of
13       Thunk f =>
14       (let
15          val v = f ()
16        in
17          t := Value v ;
18          v
19        end
20        handle e =>
21               (t := Exception e;
22                raise e)
23       )
24     | Value v => v
25     | Exception e => raise e
26 fun delay t = lazy (fn _ => force (t ()))
27 end
```

Det praktiske

- Oversættere.
- Projektfiler.
- Biblioteker.

Det praktiske: Oversættere (1/4)

Moscow ML (`http://www.itu.dk/~sestoft/mosml.html`)

- Har I arbejdet med før.
- Har en interaktiv fortolker.
- Egner sig ikke så godt til programmer som fordeler sig over flere filer.
- Implementerer ikke hele standardbiblioteket.

Det praktiske: Oversættere (2/4)

SML of New Jersey (`http://www.smlnj.org`)

- Har en interaktiv fortolker.
- Er godt dokumenteret.
- Understøtter projektfiler (CM).
- Har adskillige ekstrabiblioteker.

Det praktiske: Oversættere (3/4)

MLKit (`http://www.it-c.dk/research/mlkit`)

- Har en konfigurerbar spildopsamler.
- Understøtter projektfiler (MLB).
- Har ingen interaktiv fortolker.

Det praktiske: Oversættere (4/4)

MLTon (`http://mlton.org`)

- Har mange ekstrabiblioteker.
- Genererer meget hurtige programmer.
- Er godt dokumenteret.
- Understøtter projektfiler (MLB).
- Oversætter enormt langsomt.
- Har ingen interaktiv fortolker.

Det praktiske: Projektfiler (1/2)

Compilation Manager (`http://www.smlnj.org/doc/CM`)

- Mange avancerede funktioner.
- Godt dokumenteret.
- Kun brugt af SML of New Jersey.
- Kan være svært at sætte sig ind i.

Det praktiske: Projektfiler (2/2)

MLB: ML Basis (`http://mlton.org/MLBasis`, `http://www.it-c.dk/research/mlkit/index.php/ML_Basis_Files`)

- Brugt af flere oversættere (MLTon, MLKit, . . .).
- Ikke så avanceret som CM.
- Let at bruge og sætte sig ind i.

MyFaceTube.mlb

```
1  $(SML_LIB)/basis/basis.mlb
2
3  ORDNING.sig
4  ORDNET_MAENGDE.sig OrdnetMaengdeFn.sml
5  ORDNET_OPSLAG.sig OrdnetOpslagFn.sml
6
7  PERSON.sig Person.sml
8
9  PersonMaengde.sml
10 PersonOpslag.sml
11
12 MY_FACE_TUBE.sig MyFaceTube.sml
```

Det praktiske: Biblioteker (1/4)

The Standard ML Basis Library

(`http://www.standardml.org/Basis`)

SML's standard bibliotek. Lær det godt at kende!

Det praktiske: Biblioteker (2/4)

SMLServer (`http://www.smlserver.org`)

Web server plugin til Apache. Skriv websider med SML. Understøtter også ML Server Pages.

Det praktiske: Biblioteker (3/4)

MyLib (`http://www.github.com/mortenbp/mylib`)

Generisk bibliotek udviklet af Jesper Reenberg og mig selv. Udvider standardbiblioteket og implementerer forskellige datastrukturer og andre nyttigheder. Er i øjeblikket ikke særligt godt dokumenteret, og ændres jævnligt. Brug på eget ansvar!

Det praktiske: Biblioteker (4/4)

MLTon Lib (`http://mlton.org/cgi-bin/viewsvn.cgi/mltonlib`)

Udvidelse af standardbiblioteket samt adskillige selvindeholdte biblioteker. Svært at gå til og indeholder mange vanvittigt avancerede ting. Til gengæld er flere dele dokumenteret i videnskabelige artikler. (Kig efter Vesa Karvonen.)

Afrunding

- Minimer nødvendigheden af (implicitte?) konventioner og invarianter mellem adskilte dele af systemet.
- Find frem til hvilke dele af systemet der har med repræsentation og hvilke der har med manipulation at gøre.
- Brug projektfiler.
- Brug biblioteker (søg på Google — man bliver overrasket over hvad folk har lavet).
- Brug en eller anden form for versionskontrol.
- Vær ikke bange for at lave ting om (det kommer I alligevel til).

Afrunding: Litteratur (1/4)

Oversættere

- **MLTon**: (Anbefalet)

  `http://mlton.org`

- **SML of New Jersey**: (Anbefalet)

  `http://mlton.org`

- **Moscow ML**:

  `http://www.it-c.dk/research/mlkit`

- **MLKit**:

  `http://www.itu.dk/~sestoft/mosml.html`

Afrunding: Litteratur (2/4)

Biblioteker

- **The Standard ML Basis Library**: (Anbefalet)

  `http://www.standardml.org/Basis`

- **SMLServer**:

  `http://www.smlserver.org`

- **MyLib**:

  `http://www.github.com/mortenbp/mylib`

- **MLTon Lib**:

  `http://mlton.org/cgi-bin/viewsvn.cgi/mltonlib`

Afrunding: Litteratur (3/4)

Dokumentation

- **ML Basis**:

  `http://mlton.org/MLBasis`,

  `http://www.it-c.dk/research/mlkit/index.php/ML_Basis_Files`

- **Compilation Manager**:

  `http://www.smlnj.org/doc/CM`

- **MLLex**: (Lexer)

  `http://mlton.org/pages/Documentation/attachments/mllex.pdf`

- **MLYacc**: (Parser)

  `http://mlton.org/pages/Documentation/attachments/mlyacc.pdf`

Afrunding: Litteratur (4/4)

Bøger

- **Purely Functional Data Structures**: (Anbefalet)
  Chris Okasaki, ISBN 978-0-521-63124-2
- **Introduction to Programming Using SML**:
  Michael R. Hansen og *Hans Rischel*, ISBN 0-201-39820
- **ML For the Working Programmer**:
  Lawrence C. Paulson, ISBN 0-521-56543-X
- **The Definition of Standard ML (Revised)**:
  Robin Milner, *Mads Tofte*, *Robert Harper* og *David MacQueen*, ISBN 0-262-63181-4

KISS