



## ML (kap 5 og 6)

- Variable i ML
- Nye datatyper
- Currying
- Avanserte listeoperatorer
- Typeanalyse

INF3110/4110

INF3110/4110

## Variable

Man *kan* bruke variable i ML

```
- val x = ref 1;
val x = ref 1 : int ref
- x := 3*(!x)+5;
val it = () : unit
- !x;
val it = 8 : int
- x;
val it = ref 8 : int ref
```

Notasjonen er:

**ref** *x* oppretter en variabel (kalt «referanse») med gitt initialverdi.

**!x** gir verdien i variabelen *x*.

**r := v** legger verdien *v* i variabelen *x*.

Et eksempel på at variable er nyttige, er beregning av Fibonacci-numre.

```
- fun fib (n) = if n <= 1 then 1 else fib(n-1)+fib(n-2);
val fib = fn : int -> int
- fib(20);
val it = 10946 : int
- fib(40);
val it = 165580141 : int
- fun fib2(n) = let val f0 = ref 1 and f1 = ref 1 and i = ref 1 in
  while i < n do (
    let val temp = !f0 in
      f0 := !f1; f1 := !f1+temp; i := (i)+1
    );
  !f1
end;
val fib2 = fn : int -> int
- fib2(20);
val it = 10946 : int
- fib2(40);
val it = 165580141 : int
```

**Nye datatyper**  
Man kan definere nye datatyper i ML:

```
- datatype colour = RED | BLUE | GREEN;
datatype colour = BLUE | GREEN | RED;
- val grass = GREEN;
val grass = GREEN : colour
- grass = BLUE;
val it = false : bool
- fun norsk BLUE = "blå" | norsk GREEN = "grønn" | norsk RED = "rød",
  val norsk fn : colour -> string
- norsk GREEN;
val it = "grønn" : string
```

Noen liker å skrive konstruktorene med store bokstaver.

INF3110/4110

INF3110/4110

Man kan også bruke rekursive definisjoner av datatyper:

```
- datatype tre = BLAD of int | NODE of (tre*tre);
datatype tre = BLAD of int | NODE of tre * tre
- val busk = NODE(NODE(BLAD(3),BLAD(7)),BLAD(~4));
val busk = NODE(NODE(BLAD #,BLAD #),BLAD ~4) : tre
```

Man kan også lage funksjoner av disse datatypene

```
- fun iTree (x, BLAD(y)) = x=y
- iTree(x, NODE(a,b)) = iTree(x,a) orelse iTree(x,b);
val iTree = fn : int * int -> bool
- iTree(l, busk);
val it = false : bool
- iTree(l7, busk);
val it = true : bool
```

INF3110/4110

©Dag Langmyhr, IfI,UiO: Forelesning 25. oktober 2004

Ark 6 av 22

## Fire spesielle funksjoner

De fleste funksjonelle språk har fire nyttige høynivåfunksjoner.

### map

Denne funksjonen appliserer en funksjon på elementene i en liste.

```
- val data = [3,~17,4,8,~6] : int list
- fun double x = 2*x;
val double = fn : int -> int
- map double data;
val it = [6,~34,8,16,~12] : int list
- fun mymap f nil = nil / mymap f (x::r) = f(x)::(mymap f r);
val mymap = fn : (a -> 'b) -> 'a list -> 'b list
- mymap double data;
val it = [6,~34,8,16,~12] : int list
```

INF3110/4110

©Dag Langmyhr, IfI,UiO: Forelesning 25. oktober 2004

Ark 8 av 22

## Avansert bruk av funksjoner

Følgende er to alternativer måter å se på en funksjon med to parametere på:

$$\begin{aligned} f &: \text{int} \times \text{int} \rightarrow \text{int} \\ g &: \text{int} \rightarrow (\text{int} \rightarrow \text{int}) \end{aligned}$$

Fordelen med den siste formen er at den kan evalueres delvis:

$$g(5)$$

Dette kalles *curry-ing* etter logikeren Haskell Curry.<sup>†</sup>

```
- fun sum (a,b) = a + b;
val sum = fn : int * int -> int
- sum(3,12);
val it = 15 : int
- fun sumx a b = a + b;
val sumx = fn : int -> int -> int
- sumx 3 12;
val it = 15 : int
- val sum5 = sumx 5;
val sum5 = fn : int -> int
- sum5 ~2;
val it = 3 : int
```

<sup>†</sup> Egentlig var det Moses Schönfinkel som fant på dette først.

INF3110/4110

©Dag Langmyhr, IfI,UiO: Forelesning 25. oktober 2004

Ark 7 av 22

Man kan tenke på verdier av datatyper som funksjonskall som ikke evalueres.

Man kan tenke på verdier av datatyper som funksjonskall som ikke evalueres.

En datatype kan også inneholde data av andre typer:

```
- datatype student = BACHELOR of string / MASTER of string * real;
datatype student = BACHELOR of string / MASTER of string * real
- val per = BACHELOR("Per Hansen");
val per = BACHELOR "Per Hansen" : student
- val kari = MASTER("Kari Holm",1.3);
val kari = MASTER ("Kari Holm",1.3) : student
- fun navnBACHELOR(n) = n / navn(MASTER(n,d)) = n;
val navn = fn : student -> string
- navn kari;
val it = "Kari Holm" : string
```

INF3110/4110

## fold

Denne funksjonen «setter inn» en operator mellom elementene i en liste slik at

```
fold(+, [a1, a2, a3, ..., an]) = a1 + a2 + a3 + ... + an
```

```
- val data = [3,~17,4,8,~6];
val data = [3,~17,4,8,~6];
- fun sum (x,y) = x+y;
val sum = fn : int * int -> int;
- fun fold f nil = 0 | fold f (x::nil) = x | fold f (x::r) = f(x,fold f r);
val fold = fn : (int * int -> int) -> int list -> int
- fold sum data;
val it = ~8 : int
- fold (fn (a,b) => a*b) data;
val it = 9792 : int
- fun max2 (x,y) = if x>y then x else y;
val max2 = fn : int * int -> int
- fold max2 data;
val it = 8 : int
-
```

## filter

Denne funksjonen plukker ut de elementene i en liste som tilfredsstiller visse krav.

```
- val data = [3,~17,4,8,~6];
val data = [3,~17,4,8,~6];
- fun even x = x mod 2 = 0;
val even = fn : int -> bool
- fun filter f nil = nil | filter f (x::r) = if f(x) then x::(filter f r) else filter f r;
val filter = fn : ('a -> bool) -> 'a list -> 'a list
- filter even data;
val it = [4,8,~6] : int list
- filter (fn x => x>=0) data;
val it = [3,4,8] : int list
```

## Sammensetning

Man kan slå to funksjoner sammen til én:

```
- fun double (x) = 2*x;
val double = fn : int -> int
- fun incr5 (x) = x+5;
val incr5 = fn : int -> int
- val f = double o incr5;
val f = fn : int -> int
- f(1);
val it = 12 : int
- val g = incr5 o double;
val g = fn : int -> int
- g(1);
val it = 7 : int
- val times8 = double o double o double;
val times8 = fn : int -> int
- times8(10);
val it = 80 : int
```

## Typer

Hvorfor har vi typer?

- For å skille mellom funksjoner:  
3+4      3.0+4.0
- For å sikre mot programmeringsfeil:  
'a' + 3.14
- For å strukturere programmeringen:  
Dato iDag = new Dato(25, 10, 2004);

## Ulike arter typing i språk

### Utypete språk

Assemblerspråk/maskinspråk er de eneste utypete språk:

```
ld.w    R1, var1  
add.s   R1, #1
```

### Typesikkerhet

Noe språk mangler *typesikkerhet*, for eksempel C:

```
short endian = 1;  
if (*(char*)&endian == 0) ...  
  
printf("%d", 3.14);
```

INF3110/4110

Andre språk er «nesten sikre», som Pascal:

```
int *p;  
new(p);  
*p = 3;  
free(p);  
*p = 5;
```

Men mange språk er heldigvis typesikre: ML, Java, Perl, ...

Felles for disse er at de har *søppeltømming*.

INF3110/4110

## Typesjekking under kompilering og kjøring

Noen språk har *statisk typing* (er sterkt typet); da er alle typer kjent under kompileringen. Eksempler er ML, C og Pascal.

Fordeler:

- Raskere kode
- Finner *alle* typefeil

Andre språk har *dynamisk typing* der typen først er kjent under kjøringen; eksempler er Lisp og Perl.

De øvrige har en mellomting: Java er statisk typet med unntak av sub-klasser (og array-indekser).

INF3110/4110

## Typeanalyse

De fleste språk er basert på ren «bottom up»-analyse av typer:

```
a = f(3.14, g("txt", 17, p-2));
```

Noen tillater *overlasting* som Java:

```
int f () { ... }  
int f (int x) { ... }  
int f (Dato x) { ... }
```

Noen (som Ada) tillater også å la returtypen spille inn:

```
function f (int a): int ...  
function f (int a): char ...
```

## Verste tabbe

I Algol-60 er

$$i \wedge n \begin{cases} \text{int om } n \geq 0 \\ \text{real om } n < 0 \end{cases}$$

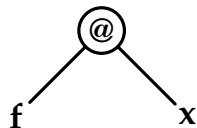
INF3110/4110

## Typeanalyse i ML

ML er i stand til å foreta en avansert analyse av uttrykkene og finne ut hvilken type de må være. Dette forklares enklest med et par eksempler.

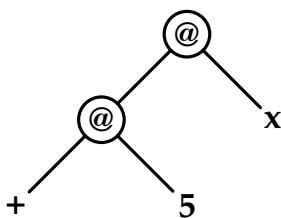
Først litt om notasjonen brukt i læreboken:

- Et funksjonskall  $f(x)$  tegnes slik:



- Infiksoperatorer angis med currying:

$$5 + x \equiv +5(x)$$

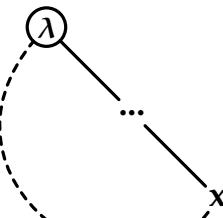


**INF3110/4110**

- Funksjoner  $fn(x) = \dots x \dots$  angis med lambdauttrykk

$$\lambda x.(\dots x\dots)$$

tegnes slik:



med angivelse av hvor variablen  $x$  er deklarert.

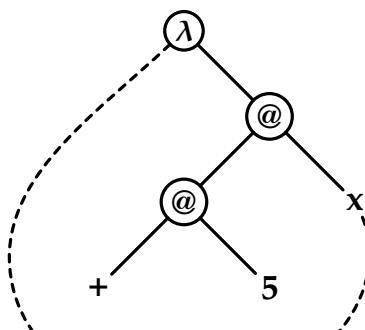
**INF3110/4110**

## Funksjon med parameter

Anta at vi har definisjonen

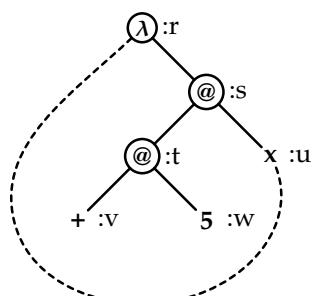
```
- fun g(x) = 5 + x;
val g = fn : int -> int
```

Tegn parseringstreet:



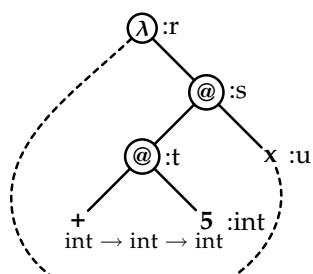
**INF3110/4110**

Sett på typenavn på alle nodene:



Så kan vi begynne å utlede:

$$\begin{array}{c} 5 \quad \text{int} \\ + \quad \text{int} \rightarrow \text{int} \rightarrow \text{int} \end{array}$$



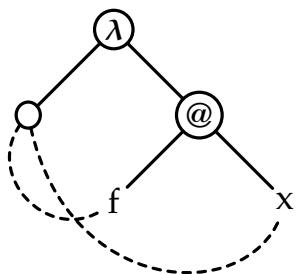
**INF3110/4110**

## En polymorf funksjon

Anta at vi har denne funksjonen

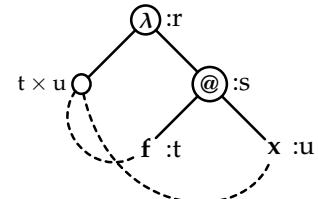
```
- fun apply (f, x) = f(x);  
val apply = fn : ('a -> 'b) * 'a -> 'b
```

Den kan tegnes slik:



INF3110/4110

Vi kan nå foreta typeanalysen:



Utledning

$$\begin{aligned} t &\equiv & u \rightarrow s \\ r &\equiv & t \times u \rightarrow s \\ &\equiv & (u \rightarrow s) \times u \rightarrow s \\ u &\equiv & a' \\ s &\equiv & b' \end{aligned}$$

INF3110/4110