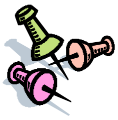
Funktionale Programmierung mit ML



Wichtigsten Eigenschaften von SML

- funktionale Programmiersprache
- interaktive Programmentwickung durch inkrementellen Übersetzer
- strenge und statische Typisierung
- strikte Auswertung von Ausdrücken (mit Einschränkungen)
- leistungsfähiges Typinferenzsystem (Typangaben meist redundant)
- Polymorphie bei Typen und Funktionen
- Zugriff auf Datenstrukturen per Pattern-matching oder Selektoren
- benutzergesteuerte Behandlung von Laufzeitfehlern
- umfangreiches Modulkonzept

Bezeichner und Operatoren

- alphabetische Bezeichnern: beginnen mit einem Buchstaben
- symbolischen Bezeichner: beliebige Folge von Zeichen
  Die symbolischen Zeichenfolgen : _ | = => -> und # dürfen nicht als Bezeichner verwendet werden.

- built-in Operatoren: *, +, <, >, ...
- Definition von Operatoren:

```
infix op1;
infixr op2;        (* rechtsassoziativer Operator *)
```

  inkl. Angabe einer Präzedenz

```
infix 30 op1;
```

- Ändern von Operatoren

```
op op1;            (* Aufheben eines Infix-Status *)
nonfix op1;        (* Löschen eines Infix-Status *)
```

Ausdrücke

- einfache Ausdrücke,
- bedingte Ausdrücke (nicht strikt)
  if B then E else E' = E , falls B den Wert true hat
  E' , sonst
- Funktionsausdrücke
  fn <Bezeichner> => <Ausdruck>
  Bsp. 1 - fn x => 2*x;
  val it = fn : int -> int
  Bsp. 2 - fn f => (fn g => (fn x => f(g(x))));
  val it = fn : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b

Anm. fn x =>e $\equiv$ λx.e

Werte

val <Bezeichner> = <Ausdruck>

Bsp. - val sekunden = 60;
val sekunden = 60 : int
- val add = fn x => (fn y => x + y); (* *)
val add = fn : int -> int -> int
- it;
val it = () : unit

simultane Deklaration: - val $(x_1,...,x_n) = (e_1,...,e_n)$
lokale Deklaration: ... let val x=2 in x*y end;

abgekürzte Schreibweise für (* *): fun add x y = x + y;
allgemein: fun <Bezeichner><formale Parameter> = <Ausdruck>

Typisierung

Eine Programmiersprache heißt **statisch typisiert**, falls alle oder die meisten Typüberprüfungen zur Übersetzungszeit durchgeführt werden können. Werden die Typprüfungen während der Laufzeit des Programms durchgeführt, so spricht man von dynamischer Typisierung.

Eine Programmiersprache heißt **streng typisiert**, wenn in jedem Programm der Sprache alle Größen zu genau einem Datentyp gehören, andernfalls schwach typisiert.

Bsp. (erlaubt in schwach typisierten Sprache, aber nicht in streng)
- sin(7) (*sin: real-> real, aber 7 ein int-Typ hat *)
- x + true (*true gehört weder zu int noch zu real *)
- 17 * 12.5 (*die Multiplikation ist entweder auf int oder auf real def. ist *)

Funktionen

<u>fun</u> <Bezeichner><formale Parameter> = <Ausdruck>

Bsp.

```
- fun ggT a b = if a=b then a else   if a<b then ggT (b-a) a else ggT (a-b) b;
  val ggT = fn : int -> int -> int
- ggT 3 9;
   val it = 3 : int

- ggT 1.0 3.0;
  stdIn:18.1-18.12 Error: operator and operand don't agree [tycon mismatch]
  operator domain: int
  operand:        real
  in expression:
    ggT 1.0

- fun ggT (a,b) = if a=b then a else if a<b then ggT (b-a,a) else ggT (a-b,b);
   val ggT = fn : int * int -> int
```


Ordnung von Funktionen

- Daten sind nullstellige Funktionen der Ordnung 0 (Konstanten)
  Bsp. 4
- Die Ordnung einer Funktion ist das Maximum der Ordnungen ihrer Argumente zuzüglich 1
  Bsp. - map (fn x => x*x) [1,2,3,4];
  val it = [1,4,9,16]:int list

Funktionen der Ordnung ≥2 heißen auch Funktionale.

Exkurs: Substitutionsstrategien

Sei f(E) die Anwendung von f auf E

a) **Call-by-value**: werte zunächst E aus und ersetze dann den formalen Parameter im Rumpf durch das Ergebnis:

Bsp. 1 <u>function</u> null x:int→int≡0
null(d(d(d(3)))) ⇒ null(d(d(3+3))) ⇒ null(d(d(6))) ⇒ ...
⇒ null(12+12) ⇒ null(24) ⇒ 0

Bsp 2: <u>function</u> fak x:nat → nat ≡ if (x=0, 1, x*fak(x-1))
fak 0 ⇒ if(0=0, 1, 0*fak(-1) ⇒ if(true, 1, 0*fak(-2))) ⇒ 

b) **call-by-name**: ersetze x überall im Rumpf textuell durch E und werte dann den Ergebnisausdruck aus.

Bsp. 3 function d x: int → int ≡ x+x
d(d(d(3))) ⇒ d(d(3)) + d(d(3)) ⇒ d(3) +d(3) + d(d(3)) ⇒ ...

c) **call-by-need** (lazy evluation): wie call-by-name, jedoch wenn E erstmalig ausgewertet wird, wird E im Rumpf überall durch den Ergebniswert ersetzt.

Bsp. 3a d(d(d(3))) ⇒ [d(d(3))+d(d(3))] ⇒ [[d(3)+d(3)] + d(d(3))] ⇒
[[[3+3]+d(3)]+d(d(3))] ⇒ [[6+6]+d(d(3))] ⇒ [12+12] ⇒ 24