Varianten (ja die Dinger heißen so)

```
type figur = Kreis of point * float
           | Rechteck of point * point
           | Dreieck of point * point * point
let flaeche = function
          Kreis(m,r) -> r *. r *. 4.*.atan 1.
        | Rechteck(p1,p2) ->
                abs_float((p1.x -. p2.x)*.(p1.y -.  p2.y))
        | Dreieck(p1,p2,p3) ->
                let ux = p1.x -. p2.x in
                let uy = p1.y -. p2.y in
                let vx = p3.x -. p2.x in
                let vy = p3.y -. p3.y in
                abs_float((ux *. vy -. vx *. uy) /. 2.)
```

Benutzung

```
val flaeche : figur -> float = <fun>
# let a = {x=0.; y=0.};;
val a : point = {x = 0.; y = 0.}
# let b = {x=1.; y=1.};;
val b : point = {x = 1.; y = 1.}
# let c = {x=0.; y=1.};;
val c : point = {x = 0.; y = 2.}
# flaeche (Dreieck(a,b,c));;
- : float = 0.5
# let k = Kreis(a, 2.0);;
val k : figur = Kreis ({x = 0.; y = 0.}, 2.)
# flaeche k;;
- : float = 12.5663706144
#
```

Allgemein

Konstruktoren sind wie Bezeichner definiert, beginnen aber mit einem Großbuchstaben. Bsp.: `Kreis`, `Dreieck`, `Datum`, `X`, `Zz_`

Ist t ein Bezeichner und sind $F_1, \ldots, F_n$ Konstruktoren und sind $typ_1, \ldots, typ_n$ Typen, so wird durch

$$\texttt{type } t \texttt{ = } F_1 \texttt{ of } typ_1 \texttt{|} \ldots \texttt{|} F_n \texttt{ of } typ_n$$

ein neuer Typ t definiert, dessen Werte von der Form $F_i(w_i)$ sind, wobei $i \in \{1, \ldots, n\}$ und w_i ein Wert des Typs typ_i ist.

Beispiel: `type t = A of int | B of int`

Die Werte des Typs `t` sind
$\{\mathtt{A}(0), \mathtt{B}(0), \mathtt{A}(1), \mathtt{B}(1), \mathtt{A}(-1), \mathtt{B}(-1), \mathtt{A}(2), \mathtt{B}(-2), \mathtt{A}(3), \mathtt{B}(-3), \ldots\}$.

"Mathematisch" ist das dasselbe, wie
$\{(0,0), (1,0), (0,1), (1,1), (0,-1), (1,-1), \ldots\}$.

Der Typ t heißt *Variante* (en.: *variant*). Die F_i sind die *Konstruktoren von* t.

Zusätzlicher Sonderfall

Es gibt auch Konstruktoren ohne Argumente:

```
type t = A of int | B of int | C
```

Hier sind die Werte
$\{\mathsf{C}, \mathsf{A}(0), \mathsf{B}(0), \mathsf{A}(1), \mathsf{B}(1), \mathsf{A}(-1), \mathsf{B}(-1), \mathsf{A}(2), \mathsf{B}(-2), \mathsf{A}(3), \mathsf{B}(-3), \ldots\}$.
Oder "mathematisch"
$\{(2,0), (0,0), (1,0), (0,1), (1,1), (0,-1), (1,-1), \ldots\}$.

Weitere Beispiele:

```
type farbe = Rot | Gruen | Blau | RGB of int*int*int
type traffic_light = Red | Amber | Green
```

Eine Variante mit ausschließlich konstanten Konstrukten (wie `traffic_light`) heißt *Aufzählungstyp*.

Verwendung von Varianten

Ist in der Variante t ein Konstruktor F `of` *typ* vorhanden so hat man folgende Typisierungsregel:

$$\Gamma \triangleright e : \mathit{typ} \quad \vdash \quad \Gamma \triangleright F\ e : t$$

Ist p ein Muster, so ist auch Fp ein Muster. Ein Wert w passt auf das Muster Fp, wenn gilt $w = F(w')$ und w' auf p passt. Liefert dieser Vergleich Ergebnis U, so ist U das Ergebnis des Vergleichs von w mit $F(w)$.

Beispiel: der Wert $\mathsf{RGB}(181, 158, 12)$ passt auf das Muster $\mathsf{RGB}(r, g, \_)$. Ergebnis ist $\{<\mathsf{r}, 181>, <\mathsf{g}, 158>\}$.

Listen

Ist A eine Menge, so ist A **list** $= \bigcup_{n \geq 0} A^n$ die Menge der *Listen* über A.

Fasst man A als Alphabet auf, so ist A **list** $= A^*$.

Man notiert Listen als $[a_1; \ldots; a_n]$ anstatt $(a_1, \ldots, a_n)$ oder $a_1 \cdots a_n$.

Die leere Liste wird mit **nil** oder $[]$ bezeichnet.

Ist $a \in A$ und $l = [a_1; \ldots; a_n] \in A$ **list**, so bezeichnet **cons**(a, l) die Liste $[a; a_1; \ldots; a_n]$.

Man schreibt auch $a :: l$ für **cons**(a, l).

Die Verkettung von Listen l_1 und l_2 schreiben wir $l_1 \mathbin{@} l_2$. Es gilt

$$[] \mathbin{@} l_2 = l_2$$
$$(a :: l) \mathbin{@} l_2 = a :: (l \mathbin{@} l_2)$$

Beispiele

```
# let x = 17::[];;
val x : int list = [17]
# let y = 9 :: 2 :: x;;
val y : int list = [9; 2; 17]
# x = y;;
- : bool = false
# x @ y;;
- : int list = [17; 9; 2; 17]
# [x;y];;
- : int list list = [[17]; [9; 2; 17]]
# [(9,2); (3,5)];;
- : (int * int) list = [(9, 2); (3, 5)]
# [2;true];;
Characters 3-7:
  [2;true];;
     ^^^^
This expression has type bool but is here used with type int
```