III.3 Ausdrücke

Mittwoch, 7. Januar 2015 08:30

Präsenzübung:

- Einsicht: 13.1.
- Bespr. in Globalübg am 14.1.
- Rückgabe in Tutorien

Ü9

- Abgabe heute
- Globalübg. heute

Evaluation:

- Tablet statt OHP
- Video-Aufzeichnung

---

plus :: Int -> (Int -> Int)

plus 2
ist Fkt. v. Typ
Int -> Int, die
Zahlen um 2
erhöht.

---

roots a b c
berechnet Nullstellen
von

$a x^2 + b x + c$

Ausdrücke:

- muss typkorrekt sein
- nach Typüberprüfung wird er ausgewertet
- nur Typüberprüfung durch :t Ausdruck

Syntaxdiagr. f. Ausdrücke.
Gib für jeden Ausdruck Typ an u. wozu er ausgewertet wird.

- Var: werden durch Strings gebildet, die mit Kleinbuchst. beginnen.
  Bsp: x, Square
- Constr: Datenkonstruktor zur Bildg. v. Objekten von Datenstrukturen. Strings mit Großbuchst. Werden nicht weiter ausgewertet.
  Bsp: True, [], :
- integer: 0, 1, -1, ...

Vordef. Typ Int

- float: -2.5, 3.4e-23 ≙ $3{,}4 \cdot 10^{-23}$
- char: 'a', '\n', ... für "newline"
- [exp$_1$, ..., exp$_n$], $n \geq 0$ steht für die Liste der n Teilausdrücke. Wenn alle exp$_i$ den Typ T haben, hat Gesamtausdruck Typ [T]. Ist abkürzende Schreibweise für exp$_1$: exp$_2$: ... : []
- String sind Listen von char. ['h', 'i'] Typ [Char] = String

Kurzschreibweise: "hi" == ['h', 'i']

- $(exp_1, \ldots, exp_n)$, $n \geq 0$

Tupel von Ausdrücken. Teilausdrücke können verschiedene Typen haben.
z.B. (10, False) hat Typ (Int, Bool)

Ausdruck () hat Typ ().

Ausdruck (exp) ist identisch zu exp.

- $(exp_1 \ldots exp_n)$, $n \geq 2$ bedeutet Funktionsanwendung d.h.:

$(((exp_1\ exp_2)\ exp_3) \ldots e_n)$

Bsp: Int -> Int

((plus 2) 3)

Int -> Int -> Int

- if expr then exp_2 else exp_3

↑ gleicher Typ

↗ Typ Bool

Wertet zuerst exp_1 aus. Davon abhängig

ergibt sich exp2 oder exp3.

- let: Alternative zu lokalen Deklar. mit where
- Lambda-Ausdrücke

\

\pat1 ... patn -> exp

steht für die Funktion, die n Argumente pat1, ..., patn auf das Ergebnis exp abbildet.

Bsp: \x -> x * 2
hat Typ Int -> Int und steht für die Funktion, die Arg. x bekommt und x*2 zurückliefert.

(\x -> x * 2) 5 =
5 * 2 =
10

(\x y -> x + y) 23 =
2 + 3 = 5

Wenn pat_i Typ T_i hat, u. exp Typ T hat, dann hat

\pat1 ... patn -> exp

Typ T1 -> ... -> Tn -> T.

Bsp:

\(x,y) -> x * y

```
\(x:xs)->x
```

```
(\(x:xs)->x) "hi" =
'h'
```

```
plus::Int->Int->Int
plus x = \y->x+y
```

Fkt, die Wert einer anderen Fkt. an Stelle 0 liefert:

```
w f = f 0
```

Dann wertet

```
w (\x->x+1)
```

aus zu

```
(\x->x+1) 0 =
0+1 =
```

λ-Ausdrücke dienen zur Def. von "anonymen" Funktionen die nur an einer bestimmten Stelle gebraucht werden.