1 Syntax

```
<Aus> ::= <Konst> | <Bez> | (<Aus>) | (<Aus>, ..., <Aus>) | #<ganze pos. Zahl><Aus>
        | <einst. Op> <Aus> | <Aus> <zweist. Op> <Aus>
        | if <Aus> then <Aus> else <Aus> | <Aus> <Aus> | let <Prog> in <Aus> end

<Typ> ::= int | real | bool | unit | (<Typ>) | (<Typ> * ... * <Typ>) | <Typ> -> <Typ>

<Dek> ::= val <Val-Muster> = <Aus>
        | fun <Bez><Arg-Muster> = <Aus>  | fun <Bez><Arg-Muster> : <Typ> = <Aus>

<Val-Muster> ::= <Bez> | (<Bez>, ... , <Bez>)

<Arg-Muster> ::= <Bez> : <Typ> | (<Bez>:<Typ>, ..., <Bez>:<Typ>)

<Prog> ::= <Dek> ... <Dek>
```

2 Environment

Function from identifiers to types or values. Composition of environments $f + g$ overwrites bindings in f.

$$(f+g)(x) = \begin{cases} g(x), & \text{if } x \in dom(g) \\ f(x), & \text{otherwise} \end{cases}$$

3 Typing

`T |- e : t` denotes that in the type environment `T` the expression `e` has the type `t`.

`T |> p : T1` denotes that in the type environment `T` the typing of the program `p` results in the type environment `T1`.

3.1 Expressions

```
T(b) = t      k \in {true, false}      k \in ZZ        k \in RR
----------    -------------------      ------------    ------------    --------------
T |- b : t    T |- k : bool            T |- k : int    T|- k : real    T |- () : unit

T |- e : t      T |- e1 : t1  ...  T |- eN : tN        T |- e : t1 * ... * tN   I \in 1..N
------------    ---------------------------------     ----------------------------------
T |- (e) : t    T |- (e1, ..., eN) : t1 * ... * tN     T |- #Ie : tI

T |- o : t1 -> t2   T |- e : t1        T |- e1 : t1   T |- o : t1 * t2 -> t   T |- e2 : t2
-------------------------------        ------------------------------------------------
T |- o e : t2                          T |- e1 o e2 : t

T |- e1 : bool   T |- e2 : t   T |- e3 : t      T |- e1 : t1 -> t2   T |- e2 : t1
-----------------------------------------      --------------------------------
T |- if e1 then e2 else e3 : t                 T |- e1 e2 : t2

T |> p : T1   T1 |- e : t
-------------------------
T |- let p in e end : t
```

3.2 Declarations

```
T |- e : t                            T + [f := t1 -> t2] + [b := t1] |- e : t2
---------------------------           ------------------------------------------------
T |> val b = e : T + [b := t]         T |> fun f (b : t1) : t2 = e : T + [f := t1 -> t2]
```

3.3 Programs

```
T0 |> d1 : T1  ...  TN |> dN : T(N+1)
-------------------------------------
T0 |> d1 ... dN : T(N+1)
```