1 Syntax

```
<Exp> ::= <Const> | <Ident> | (<Exp>) |
        | <un. Op> <Exp> | <Exp> <bin. Op> <Exp>
        | if <Exp> then <Exp> else <Exp> | <Exp> <Exp> | let <Prog> in <Exp> end

<Type> ::= int | bool | (<Type>) | <Type> -> <Type>

<Dec> ::= val <Ident> = <Exp> | fun <Ident> <Ident> = <Exp>

<Prog> ::= <Dec> ... <Dec>
```

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        T |- e : t
----------    -------------------     ------------    ------------
T |- b : t    T |- k : bool           T |- k : int    T |- (e) : t

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, x : r1 |- e : t2                T |> p : T1   T1 |- e : t
-------------------------          -------------------------
T |- fn x => e : t1 -> t2          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 = e : T + [f := t1 -> t2]
```

3.3 Programs

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

4 Procedural values

```
(fun f b = e, V)
```

5 Evaluation

`V |= e : v` denotes that in the value environment `V` the expression `e` evaluates to the value `v`.

`V |>> p : V1` denotes that in the value environment `V` the evaluation of the program `p` results in the value environment `V1`.

5.1 Expressions

```
V(b) = v
----------    ----------
V |= b : v    V |= k : k

V |= e : v      V |= e : v        V |= e1 : v1   V |= e2 : v2
------------    --------------    ---------------------------
V |= (e) : v    V |= o e : o v    V |= e1 o e2 : v1 o v2

V |= e1 : true   V |= e2 : v       V |= e1 : false   V |= e3 : v
------------------------------     ------------------------------
V |= if e1 then e2 else e3 : v     V |= if e1 then e2 else e3 : v

V |= e1 : (fun f b = e, V1)   V |= e2 : v2
V1 + [f := (fun f b = e, V1)] + [b := v2] |= e : v
-----------------------------------------------------
V |= e1 e2 : v

V |>> p : V1   V1 |= e : v
--------------------------
V |= let p in e end : v
```

5.2 Declarations

```
V |= e : v
------------------------------
V |>> val b = e : V + [b := v]

V1 = (V restricted to FreeIds(fun f b = e))
------------------------------------------------------------------------
V |>> fun f b = e : V + [f := (fun f b = e, V1)]
```

5.3 Programs

```
V0 |>> d1 : V1  ...  V(N-1) |>> dN : VN
---------------------------------------
V0 |>> d1 ... dN : VN
```

5.4 Examples

```
[x:=1] |= x : 1   [x:=1] |= 3 : 3
---------------------------------
[x:=1] |= x+3 : 4

[x:=1, a:=2, f:= (fun f x = x+a, [a:=2])] |= f : (fun f x = x+a, [a:=2])
[x:=1, a:=2, f:= (fun f x = x+a, [a:=2])] |= x+3 : 4
[a:=2] + [f : (fun f x = x+a, [a:=2])] + [x:=4] |= x+a : 6
-------------------------------------------------------------------------------------------
[x:=1, a:=2, f:= (fun f x = x+a, [a:=2])] |= f (x+3) : 6

[] |>> val x = 1 : [x:=1]
[x:=1] |>> val a = 2 : [x:=1, a:=2]
[x:=1, a:=2] |>> fun f x = x+a : [x:=1, a:=2, f:= (fun f x = x+a, , [a:=2])]
[x:=1, a:=2, f:= (fun f x = x+a, [a:=2])] |>> val y = f (x+3) : ... + [y:=6]
------------------------------------------------------------------------------------------------
[] |>> val x = 1  val a = 2  fun f x = x+a  val y = f (x+3)
         : [x:=1, a:=2, f:= (fun f x = x+a, [a:=2]), y:=6]
```

6 Procedures w/o names

```
2;
```

```
it + it;
```

```
let fun f x = x+1 in f end;
```

```
it 1;
```

```
val f = let fun f (x:int) = x+1 in f end;
```

```
f 1;
```

```
fn x => x+1;
```

```
it 1;
```

```
val f = (fn x => x+1);
```

```
f 1;
```

```
val f = fn x => x+1;
```

```
f 1;
```

7 Curried procedures

```
 val add =
    (fn x =>
       (fn y =>
          x+y
       )
    );
```

```
add 1 2;
```

```
fun add x y = x+y
```

```
add 1;
```

```
it 2;
```

```
val inc = add 1;
```

```
inc 1;
```

```
(add 1) 2
```

```
int -> (int -> int)
```

8 Illustration of Tail recursion

8.1 Non tail-recursive procedure f

```
F:=(fun f x = x+f(x+1), [])
V:=[f:=F, x:=1]

                        V |= f : F      [f:=F, x:=2] |= x+f(x+1) : v2
                        V |= x+1 : 2
                        ---------------------------------------------
            V |= x : 1  V |= f(x+1) : v2  1+v2 = v1
            ---------------------------------------
V |= f : F  V |= x+f(x+1) : v1
V |= x : 1
------------------------------
V |= f x : v1
```

8.2 Tail-recursive procedure g

```
G:=(fun g x = g(x+1), [])
V:=[g:=G, x:=1]

                        V |= g : G      [g:=G, x:=2] |= g(x+1) : v1
                        V |= x+1 : 2
                        -------------------------------------------
            V |= x : 1  V |= g(x+1) : v1
            ----------------------------
V |= g : G  V |= g(x+1) : v1
V |= x : 1
----------------------------
V |= g x : v1
```