4 Procedures as triples

`(fun f M = e, t, 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 |= e1 : v1  ...  V |= eN : vN        V |= e : (v1, ..., vN)
-----------     ---------------------------------     ----------------------
V |= (e) : v    V |= (e1, ..., eN) : (v1, ..., vN)    V |= #Ie : vI

V |= e : v         V |= e1 : v1   V |= e2 : v2
--------------     --------------------------
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, t, V1)   V |= e2 : v2
V1 + [f := (fun f b = e, t, 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 : t1) : t2 = e))
-------------------------------------------------------------------------
V |>> fun f (b : t1) : t2 = e : V + [f := (fun f b = e, t1 -> t2, 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, int -> int, [a:=2])] |= f : (fun f x = x+a, int -> int, [a:=2])
[x:=1, a:=2, f:= (fun f x = x+a, int -> int, [a:=2])] |= x+3 : 4
[a:=2] + [f : (fun f x = x+1, int -> int, [a:=2])] + [x:=4] |= x+a : 5
-----------------------------------------------------------------------------------------
[x:=1, a:=2, f:= (fun f x = x+a, int -> int, [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:int):int = x+a : [x:=1, a:=2, f:= (fun f x = x+a, int -> int, [a:=2])]
[x:=1, a:=2, f:= (fun f x = x+a, int -> int, [a:=2])] |>> val y = f (x+3) : ... + [y:=6]
-------------------------------------------------------------------------------------------------
[] |>> val x = 1  val a = 2  fun f (x:int):int = x+a  val y = f (x+3)
         : [x:=1, a:=2, f:= (fun f x = x+a, int -> int, [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 (kaskadierte)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 Iterators

```
sum_seq n = 0 + 1 + ... + n

fun sum_seq (n:int) = if n < 1 then 0 else n + sum_seq (n-1)

sum_of_sqr n = 0 + 1*1 + ... + n*n

fun sum_sqr (n:int) = if n < 1 then 0 else n*n + sum_sqr (n-1)


sum f n = 0 + f 1 + ... + f n

sum_seq n = sum (fn x => x) n
sum_sqr n = sum (fn x => x*x) n

sum: (int->int) -> int -> int

fun sum (f : int->int) (n:int) = if n < 1 then 0 else sum f (n-1) + f n

sum_seq 3 = sum (fn x => x) 3

sum_of_squares 3 = sum (fn x => x*x) 3

% bounded iteration

iter n s f  = f(...(f s)...)

iter 3 s f = f(f(f(s)))

fun iter (n:int) (s:int) (f:int->int) = if n<1 then s else iter (n-1) (f s) f

fun two_power_n n = iter n 1 (fn x => 2*x)

% unbounded iteration

first s p = minimal t >= s such that p s is true

first 0 (fn x => x >= 3) = 3

fun first (s:int) (p : int -> bool) = if p s then s else first (s+1) p
```