fun f x = 
  if x > 3 (* x : int *) 
  then 42 (* returns an int *) 
  else x * 2 (* consistent! *)

(* g : ERROR *)
(* fun g x =    if x > 3    then true     else x * 2 *)

val x = 42

fun f (y, z, w) = 
  if y 
  then z + x 
  else 0

fun f x = 
  let 
    val (y, z) = x 
  in 
    (abs y) + z 
  end

fun sum xs = 
  case xs 
  of [] ⇒ 0 
    | x :: xs' ⇒ x + (sum xs')

fun sumr xs = 
  case xs 
  of [] ⇒ 0.0 
    | x :: xs' ⇒ x + (sum xs')

fun broken_sum xs = 
  case xs 
  of [] ⇒ 
    | x :: xs' ⇒ x + (broken_sum x)

(* also borked
fun broken_sum xs = 
  case xs of 
    | [] ⇒ [] 
    | x :: xs' ⇒ broken_sum x

(* sum : T1 −> T2xs : T1T1 = T3 listT2 = intT3 = intsum : int list −> intBARF : ... => x + (broken_sum x)*)
fun length xs = case xs of [] ⇒ 0 | x :: xs' ⇒ 1 + (length xs')

(*
val r = ref NONE
(* val _ = r := SOME "hi" val i = 1 + valOf (!r) *)
type α foo = α ref
val f : α → α foo = ref
val r2 = f NONE (* also need value restriction here *)
(* where the value restriction arises despite no mutation *)
val pairWithOne = List.map (λ x ⇒ (x,1))
(* a workaround *)
fun pairWithOne2 xs = List.map (λ x ⇒ (x,1)) xs

(*
val x = ref 42
val y = ref 42
val z = x
val _ = x := 43
val w = (!y) + (!z) (* 85 *)
Foo x = new Foo();
Foo y = new Foo();
x.myInt = 42;
y.myInt = 42;
Foo z = x;
x.myInt = 43;
print (y.myInt + z.myInt)
x)