fun f x = (* infer val f : int -> int *)
  if x > 3
  then 42
  else x * 2
(*
fun g x = (* report type error *)
  if x > 3
  then true
  else x * 2
  ()
val x = 42 (* val x : int *)
fun f (y, z, w) = (* y must be bool *)
  if y
    then z + x
    (* z must be int *)
  else 0
  (* both branches have same type *)
  (* I must return an int
    I must take a bool * int * ANYTHING
    so val f : bool * int * 'a -> int
  *)
(*
  f : T1 -> T2 [must be a function; all functions take one argument]
  x : T1 [must have type of f’s argument]
  y : T3
  z : T4
T1 = T3 * T4 [else pattern-match in val-binding doesn’t type-check]
T3 = int [because (abs y) where abs : int -> int]
T4 = int [because add z to an int]
So T1 = int * int
So (abs y) + z : int, so let-expression : int, so body : int, so T2=int
So f : int * int -> int
*)
fun f x = (*
    let val (y,z) = x in
    (abs y) + z
  end
(*
  sum : T1 -> T2 [must be a function; all functions take one argument]
  xs : T1 [must have type of f’s argument]
  x : T3
  y : T4
T1 = T3 * T4 [else pattern-match in val-binding doesn’t type-check]
T3 = int [because (abs y) where abs : int -> int]
T4 = int [because add z to an int]
So T1 = int * int
So (abs y) + z : int, so let-expression : int, so body : int, so T2=int
So f : int * int -> int
*)
fun sum xs = (*
    case xs of
    [] => 0
    | x::xs' => x + (sum xs')
  *)
fun length xs = (*
    length : T1 -> T2 [must be a function; all functions take one argument]
    xs : T1 [must have type of f’s argument]
    x : T3
    y : T4
T1 = T3 * T4 [else pattern-match on xs doesn’t type-check]
T3 = int [because (abs y) where abs : int -> int]
T4 = int [because add z to an int]
So T1 = int * int
So (abs y) + z : int, so let-expression : int, so body : int, so T2=int
So f : int * int -> int
*)
fun length xs = (*
    case xs of
    [] => 0
    | x::xs' => 1 + (length xs')
  *)
(*
  f : T1 * T2 * T3 -> T4
  x : T1
  y : T2
  z : T3
T1 = T3 * T4 [else pattern-match on xs doesn’t type-check]
T3 = int [because (abs y) where abs : int -> int]
T4 = int [because add z to an int]
So T1 = int * int
So (abs y) + z : int, so let-expression : int, so body : int, so T2=int
So f : int * int -> int
*)
fun f (x,y,z) = (*
    if true
    then (x,y,z)
    else (y,x,z)
  *)
fun compose (f, g) = λ x ⇒ f (g x)

(* this first line is not polymorphic so next two lines do not type-check *)

val r = ref NONE

val _ = r := SOME "hi"

val i = 1 + valOf (!r)

fun compose (f, g) = λ x ⇒ f (g x)

(val r = ref NONE)

(val _ = r := SOME "hi"

val i = 1 + valOf (!r)

(* where the value restriction arises despite no mutation *)

fun pairWithOne2 xs = List.map (λ x ⇒ (x,1)) xs

(* a workaround *)