Typing rules:

Typing rule set!:

For every: type environment \_Tenv,

variable reference \_x1

expressions \_e1 and

type expressions \_S1:

If \_Tenv |- \_e1 : \_S1 and

\_Tenv |- \_x1 : \_S1

Then \_Tenv |- (set! \_x1 \_e1) : void

Typing rule LitExp:

  For every: type environment \_Tenv,

             symbol expression \_symb1

             compound sexp \_sexp1

             number expression \_n

             boolean expression \_b

             string expession \_s

  \_Tenv |- \_symb1 : Symbol(\_symb1)

  \_Tenv |- '() : Symbol

  \_Tenv |- \_sexp1: Pair

  \_Tenv |- \_n: Number

  \_Tenv |- \_b: Boolean

  \_Tenv |- \_s: String

1. False. g accepts type **T1**, but **a** is a number. There’s no guarantee that **T1** is a number hence g(a) might not be defined.
2. False. f accepts type **T2** and **x** is **T1** hence f(x) might not be defined.
3. True. f accepts type **T1** and returns **T2**. The lambda wrapping f simply returns the value of that f returns, **x** is of type **T1** hence f(x) is of type **T2** so the lambda will return type **T2** as well
4. True. f accepts T1 X T2 variables and gets **x** and **y**. **y** is of type **T2** and **x** can be inferred because he is bound variable. The lambda returns the value of f which is **T3**
5. 1. ((lambda (f x1) (f 1 x1)) + #t) :