Question 1: Which of the following typing statement is true / false, explain why

(a) False.

in the following example, We can see that the function g should get an input of type T1. G get a as an input, and a is type number. But we cant be sure that T1 is indeed type number.

Therefore the application (f (g a)) fail because "a" is not a valid input for function g.

(b) False.

We can see that the function f should get an input of type T2.

The input of f is x which is from type T1.

If we will take T1 = Boolean and T2 = Symbol we will get that

f : [Symbol → Boolean] Therefore the application (f x) will fail because x is from type Boolean. It could work if T1 was the same as T2.

(c) True.

The lambda should return the output of f(x) which should be of type T2. Indeed, the type of the input of f is matched to the type of x - T1 and to the return type which is T2.

(d) True.

The function f is given an input of type T1 and T2.

The lambda gets input x and in its body it applicates f on x and y.

The input of the lambda is x, so x should be from type T1. X is not a free variable, so we don’t have any assumption about x's type in the TEnv on the left side. So we can infer that indeed the type of x is T1, y from type T2 which is match to the input of f (T2) . In the output of f is indeed T3 – so it's correct.

Question 2: Perform type inference manually on the following expressions, using the Type Equations method. List all the steps of the procedure

(a) ((lambda (f x1) (f 1 x1)) + #t)

Step 1- Rename bound variables**:**

((lambda (f x1) (f 1 x1)) + #t) turns - **((lambda (f x) (f 1 x)) + #t)**

Step 2- Assign type variables to all sub-exps:

Expression	Variable
((lambda (f x) (f 1 x)) + #t)	T0
(lambda (f x) (f 1 x))	T1
(f 1 x)	T2
f	Tf
1	Tnum1
x	Tx
+	T+
#t	T#t

Step 3 - Construct type equation:

Expression	Equation
((lambda (f x) (f 1 x)) + #t)	T1 = [T+ * T#t -> T0]
(lambda (f x) (f 1 x))	T1 = [Tf * Tx -> T2]
(f 1 x)	Tf= [Tnum1*Tx->T2]
1	Tnum1 = Number
+	T+ = [Number*Number->Number]
#t	T#t = Boolean

Step 4 - Solving the equation:

Expression	Substitution
1. T1 = [T+ * T#t -> T0]	{}
2. T1 = [Tf*Tx -> T2]	
3. Tf= [Tnum1*Tx->T2]	
4. Tnum1 = Number	
5. T+ = [Number * Number -> Number]	
6. T#t = Boolean	

Equation 1:

- The empty substitution is applied to Eq1.
- Eq1 is applied to the substitution.

Expression	Substitution
2. T1 = [Tf*Tx -> T2]	**{T1 := [T+ * T#t -> T0]}**
3. Tf= [Tnum1*Tx->T2]	
4. Tnum1 = Number	
5. T+ = [Number * Number -> Number]	
6. T#t = Boolean	

Equation 2:

- The substitution is applied to Eq2. [Tf * Tx -> T2] = [T+ * T#t -> T0]
- Eq2 is applied to the substitution – split into equations between corresponding components and add to the set of equations.

Expression	Substitution
3. Tf= [Tnum1*Tx->T2]	{T1 := [T+ * T#t -> T0]}
4. Tnum1 = Number	
5. T+ = [Number * Number -> Number]	
6. T#t = Boolean	
7. Tf= T+	
8. Tx= T#t	
9. T2=T0	

Equation 3:

- The substitution is applied to Eq3.no change
- Eq3 is add to the substitution.

Expression	Substitution
4. Tnum1 = Number	{T1 := [T+ * T#t -> T0], **Tf= [Tnum1*Tx->T2]}**
5. T+ = [Number * Number -> Number]	
6. T#t = Boolean	
7. Tf= T+	
8. Tx= T#t	
9. T2=T0	

Equation 4:

- The substitution is applied to Eq4. no change
- Eq4 is applied to substitution: Occurrences of Tnum1 are substituted by Number.

Expression	Substitution
5. T+ = [Number * Number -> Number]	{ T1 := [T+ * T#t -> T0], Tf= **[Number** *Tx->T2], Tnum1 = Number }
6. T#t = Boolean	
7. Tf= T+	
8. Tx= T#t	
9. T2=T0	

Equation 5:

- The sub is applied to Eq5 (no change).
- Eq5 is applied to substitution: Occurrences of T+ are substituted by : [Number * Number -> Number].

Expression	Substitution
6. T#t = Boolean	{ T1 := **[Number * Number -> Number]** * T#t -> T0], Tf= [Number *Tx->T2], Tnum1 = Number, **T+ = [Number * Number -> Number]** }
7. Tf= T+	
8. Tx= T#t	
9. T2=T0	

Equation 6:

- The substitution is applied to Eq6. no change
- Eq6 is applied to substitution.

Expression	Substitution
7. Tf= T+	{T1 := [Number * Number -> Number] * **Boolean**-> T0], Tf= [Number *Tx->T2], Tnum1 = Number, T+ = [Number * Number -> Number], **T#t = Boolean** }
8. Tx= T#t	
9. T2=T0	

Equation 7:

- in the expression Tf = T+
- in the substitution-
  Tf = [Number *Tx->T2]
  T+ = [Number * Number -> Number]
  so we get: [Number *Tx->T2] = [Number * Number -> Number]
  so we add to the equations:
  Tx = Number, T2 =Number.

Expression	Substitution
8. Tx= T#t	{T1 := [Number * Number -> Number] * Boolean -> T0], Tf= [Number *Tx->T2], Tnum1 = Number, T+ = [Number * Number -> Number], T#t = Boolean }
9. T2=T0	
10. Tx = Number	
11. T2 =Number	

Equation 8:

- The sub is applied to Eq8 : We get: Tx = Boolean
- Eq8 is applied to substitution: Occurrences of Tx are substituted by Boolean.

Expression	Substitution
9. T2=T0	{T1 := [Number * Number -> Number] * Boolean -> T0], Tf= [Number * **Boolean** ->T2]**,** Tnum1 = Number**,** T+ = [Number * Number -> Number]**,** T#t = Boolean, **Tx = Boolean** }
10. Tx = Number	
11. T2 =Number	

Equation 9:

- The sub is applied to Eq9 – no change.
- Eq8 is applied to substitution: Occurrences of T2 are substituted by T0.

Expression	Substitution
10. Tx = Number	{T1 := [Number * Number -> Number] * Boolean -> T0], Tf= [Number * Boolean ->**T0**], Tnum1 = Number, T+ = [Number * Number -> Number], T#t = Boolean, Tx = Boolean, **T2=T0** }
11. T2 =Number	

Equation 10:

- The sub is applied to Eq10 – we get: Number = Boolean.
- therefore they are conflicting equations.

Final answer - the expression not well typed

(b) ((lambda (f1 x1) (f1 x1 1)) + *)

Step 1- Rename bound variables**:**

((lambda (f1 x1) (f1 x1 1)) + *) turns: ((lambda (f x) (f x 1)) + *)

Step 2- Assign type variables to all sub-exps:

Expression	Variable
((lambda (f x) (f x 1)) + *)	T0
((lambda (f x) (f x 1))	T1
(f x 1)	T2
f	Tf
x	Tx
1	Tnum1
+	T+
*	T*

Step 3 - Construct type equation:

Expression	Equation
((lambda (f x) (f x 1)) + *)	T1 = [T+ * T* -> T0]
((lambda (f x) (f x 1))	T1 = [Tf * Tx -> T2]
(f x 1)	Tf= [Tx * Tnum1 ->T2]
1	Tnum1 = Number
+	T+ = [Number * Number -> Number]
*	T* = [Number * Number -> Number]

Step 4 - Solving the equation:

Expression	Substitution
1. T1 = [T+ * T* -> T0]	{}
2. T1 = [Tf * Tx -> T2]	
3. Tf= [Tx * Tnum1 ->T2]	
4. Tnum1 = Number	
5. T+ = [Number * Number -> Number]	
6. T* = [Number * Number -> Number]	

Equation 1:

- The empty substitution is applied to Eq1.
- Eq1 is applied to the substitution.

Expression	Substitution
2. T1 = [Tf * Tx -> T2]	**{ T1 := [T+ * T* -> T0] }**
3. Tf= [Tx * Tnum1 ->T2]	
4. Tnum1 = Number	
5. T+ = [Number * Number -> Number]	
6. T* = [Number * Number -> Number]	

Equation 2:

- The substitution is applied to Eq2. [Tf*Tx -> T2]= [T+* T* ->T0]
- Eq2 is applied to the substitution – split into equations between corresponding components and add to the set of equations.

Expression	Substitution
3. Tf= [Tx * Tnum1 ->T2]	{ T1 := [T+ * T* -> T0]}
4. Tnum1 = Number	
5. T+ = [Number * Number -> Number]	
6. T* = [Number * Number -> Number]	
7. Tf= T+	
8. Tx= T*	
9. T2=T0	

Equation 3:

- The substitution is applied to Eq3 - no change
- Eq3 is add to the substitution.

Expression	Substitution
4. Tnum1 = Number	{T1 := [T+ * T* -> T0] **Tf= [Tx*Tnum1->T2]}**
5. T+ = [Number * Number -> Number]	
6. T* = [Number * Number -> Number]	
7. **Tf= T+**	
8. **Tx= T***	
9. **T2=T0**	

Equation 4:

- The substitution is applied to Eq4. no change
- Eq4 is applied to substitution: Occurrences of Tnum1 are substituted by Number.

Expression	Substitution
5. T+ = [Number * Number -> Number]	{T1 := [T+ * T*-> T0]} {Tf= [Tx * **Number**->T2], **Tnum1 = Number**}
6. T* = [Number * Number -> Number]	
7. **Tf= T+**	
8. **Tx= T***	
9. **T2=T0**	

Equation 5:

- The sub is applied to Eq5 (no change).
- Eq5 is applied to substitution: Occurrences of T+ are substituted by :

  [Number * Number -> Number].

Expression	Substitution
6. T* = [Number * Number -> Number]	{T1 := **[[Number * Number -> Number]** * T* -> T0], Tf= [Tx * Number->T2], Tnum1 = Number, **T+ = [Number * Number -> Number]}**
7. **Tf= T+**	
8. **Tx= T***	
9. **T2=T0**	

Equation 6:.

- The substitution is applied to Eq6. no change
- Eq6 is applied to substitution: Occurrences of T* are substituted by :

  [Number * Number -> Number].

Expression	Substitution
7. Tf= T+	{T1 :=[[Number * Number -> Number] * **[Number * Number -> Number]** -> T0], Tf := [Number *Tx->T2], Tnum1 := Number, T+ := [Number * Number -> Number], **T* := [Number * Number -> Number]**}
8. Tx= T*	
9. T2=T0	

Equation 7:

- The substitution is applied to Eq7.

  we get: [Number *Tx->T2] = [Number * Number -> Number]
- We add to the equations : Tx = Number, T2 = Number.

Expression	Substitution
8. Tx= T*	{T1 :=[[Number * Number -> Number] * [Number * Number -> Number] -> T0], Tf := [Number *Tx->T2], Tnum1 := Number, T+ := [Number * Number -> Number], T* := [Number * Number -> Number]}
9. T2=T0	
10. Tx = Number	
11. T2 =Number	

Equation 8:

- The sub is applied to Eq8 :
- We get: Tx = [Number * Number -> Number]
- Eq8 is applied to substitution: Occurrences of Tx are substituted by [Number * Number -> Number]

Expression	Substitution
9. T2=T0	{T1 :=[[Number * Number -> Number] * [Number * Number -> Number] -> T0], Tf := [Number ***[Number * Number -> Number***]->T2], Tnum1 := Number, T+ := [Number * Number -> Number], T* := [Number * Number -> Number], **Tx = [Number * Number -> Number]**}
10. Tx = Number	
11. T2 =Number	

Equation 9:

- The sub is applied to Eq9 – no change.
- Eq8 is applied to substitution: Occurrences of T2 are substituted by T0.

Expression	Substitution
10. Tx = Number	{T1 :=[[Number * Number -> Number] * [Number * Number -> Number] -> T0], Tf := [**Number *[Number *** Number -> Number]->**T0**], Tnum1 := Number, T+ := [Number * Number -> Number], T* := [Number * Number -> Number], Tx = [Number * Number -> Number] **T2=T0**}
11. T2 =Number	

Equation 10:

- The sub is applied to Eq10 – we get:

- Number = = [Number * Number -> Number].
- therefore they are conflicting equations.s

Final answer - the expression <u>not well typed</u>

Typing rule

set!

For every: type environment _Tenv,
variable _x1
expressions _e1 and
type expressions _S1:
If _Tenv |- _x1: _S1 and_Tenv |- _e1: _S1
Then _Tenv |- (set! _x1 _e1) : void

Lit

For every: type environment _Tenv,
symbol expression_symb1
compound sexp_sexp1
number expression _n
Boolean expression _b
string expression _s
_Tenv |- _symb1 : Symbol(_symb1)
_Tenv |- '( ) : Symbol
_Tenv |- _sexp1 : Pair
_Tenv |- _n : Number
_Tenv |- _b : Boolean
_Tenv |- _s : String

Define-type

Define-type(typeName: string, records: Record[])
For every:
type environment _Tenv
identifiers id_i (i in [1…n]
with var decleration varDec_ij (field_ij) (i in [1...n], j in [1..R_i])
_Tenv |- (define-type id (id_1(varDec_11 … vardec_1r1)) ….) : Void

Type-case

Type-case (typeName: string, val: CExp, cases: CaseExp[])
For every: user-defined-type-id:
type environment _Tenv,
val CExp,
with component records record_1 ... record_n
with fields (field_ij) (i in [1...n], j in [1..R_i])
body_i for i in [1…n] sequences of CExp
type expressions _t1,…,_tn
type expressions _T where T covers all types _t1,…,_tn
If _Tenv |-body_i : _ti for all i in [1…n]
Then _Tenv |- (Type-case id val (record1(field_11 ... field_1r1) body_1)...): _T