PPL Assignment 4

Part 1: Theoretical Questions

1) Which of the following typing statement is true / false, explain why
   a) **false** - g receives 'a' of type $T1$ and returns a result of type $T2, f$ receives $g(x)$ of type $T2$ instead of type $T1$.
   b) **true** - f receives 'y' of type $T2$ and returns a result of type $T1$.
   c) **true** - f receives 'x' as a parameter of type $T1$ and returns a result of type $T2$.
   d) **false** - f receives 'x' and '100' as parameters, and it's false because T3 is not necessarily of type number.

2) Perform type inference manually on the following expressions, using the Type Equations method. List all the steps of the procedure
   a) ((lambda (x1) (+ x1 1)) 4)
      i) Rename bound variables:
      ((lambda (x) (+ x 1)) 4)
      ii) make type variables to every sub expression:

Expression	**Variables**
((lambda (x) (+ x 1)) 4)	T-app
(lambda (x) (+ x 1))	T-proc
(+ x 1)	T-add
+	T+
x	Tx
1	T1
4	T4

iii) make equations:

Sub-Expressions:

Expression	Equation
((lambda (x) (+ x 1)) 4)	T-proc = [T4 ⟶ T-app]
(lambda (x) (+ x 1))	T-proc = [Tx ⟶ T-add]
(+ x 1)	T+ = [Tx * T1 ⟶ T-add]

Primitive equations:

Expression	Equation
+	T+ = [number *number ⟶ number]
1	T1 = number
4	T4 = number

iv) Solve the equations:

Equation	Substitution
T-proc = [T4 ⟶ T-app]	{ }
T-proc = [Tx ⟶ T-add]	
T+ = [Tx * T1 ⟶ T-add]	
T+ = [number *number ⟶ number]	
T1 = number	
T4 = number	

Equation	Substitution
T-proc = [Tx ⟶ T-add]	{ T-proc = [T4 ⟶ T-app] }
T+ = [Tx * T1 ⟶ T-add]	
T+ = [number *number ⟶ number]	
T1 = number	
T4 = number	

Equation	**Substitution**
T+ = [Tx * T1 ⟶ T-add]	{ T-proc = [T4 ⟶ T-app] }
T+ = [number *number ⟶ number]	
T1 = number	
T4 = number	
Tx = T4	
T-app = T-add	

Equation	**Substitution**
T+ = [number *number ⟶ number]	{ T-proc = [T4 ⟶ T-app],
T1 = number	T+ = [Tx * T1 ⟶ T-add] }
T4 = number	
Tx = T4	
T-app = T-add	

Equation	**Substitution**
T1 = number	{ T-proc = [T4 ⟶ T-app],
T4 = number	T+ = [number *number ⟶ number] }
Tx = T4	
T-app = T-add	
Tx = number	
T1 = number	
T-add = number	

Equation	Substitution
Tx = T4	{ T-proc = [T4 → T-app],
T-app = T-add	T+ = [number *number → number],
Tx = number	T1 = number, T4 = number }
T-add = number	

Equation	Substitution
T-app = T-add	{ T-proc = [T4 → T-app],
Tx = number	T+ = [number *number → number],
T-add = number	T1 = number, T4 = number,
	Tx = number }

Equation	Substitution
	{ T-proc = [T4 → T-app],
	T+ = [number *number → number],
	T1 = number, T4 = number,
	Tx = number, T-add = number ,
	T-app = number }

The type inference succeeds since we have a type for T-app, meaning that the expression is well typed. Because there are no free variables, the inferred type of T-app is: **Number**.
<u>The expression is now typed as</u>: **((lambda (x1 : number) : number (+ x1 1)) 4)**

b) ((lambda (f1 x1) (f1 x1 1)) 4 +)

i) Rename bound variables:

((lambda (f x) (f x 1)) 4 +)

ii) make type variables to every sub expression:

Expression	Variables
((lambda (f x) (f x 1)) 4 +)	T0
(lambda (f x) (f x 1))	T1
(f x 1)	T2
f	Tf
x	Tx
1	Tnum1
4	Tnum4
+	T+

iii) make equations:

Sub-Expressions:

Expression	Equation
((lambda (f x) (f x 1)) 4 +)	T1 = [Tnum4 * T+ ⟶ T0]
(lambda (f x) (f x 1))	T1 = [Tf * Tx ⟶ T2]
(f x 1)	Tf = [Tx * Tnum1 ⟶ T2]

Primitive equations:

Expression	Equation
+	T+ = [number *number ⟶ number]
1	Tnum1 = number
4	Tnum4 = number

iv) Solve the equations:

Equation	**Substitution**
T+ = [number *number ⟶ number]	{ }
Tnum1 = number	
Tnum4 = number	
T1 = [Tnum4 * T+ ⟶ T0]	
T1 = [Tf * Tx ⟶ T2]	
Tf = [Tx * Tnum1 ⟶ T2]	

Equation	**Substitution**
Tnum1 = number	{ T+ = [number *number ⟶ number] }
Tnum4 = number	
T1 = [Tnum4 * T+ ⟶ T0]	
T1 = [Tf * Tx ⟶ T2]	
Tf = [Tx * Tnum1 ⟶ T2]	

Equation	**Substitution**
T1 = [Tnum4 * T+ ⟶ T0]	{ T+ = [number *number ⟶ number],
T1 = [Tf * Tx ⟶ T2]	Tnum1 = number, Tnum4 = number }
Tf = [Tx * Tnum1 ⟶ T2]	

Equation	**Substitution**
T1 = [Tf * Tx ⟶ T2]	{ T+ = [number *number ⟶ number],
Tf = [Tx * Tnum1 ⟶ T2]	Tnum1 = number, Tnum4 = number,
	T1 = [number * [number *number ⟶ number] ⟶ T0] }

Equation	**Substitution**
T1 = [Tf * Tx → T2]	{ T+ = [number *number → number],
Tf = [Tx * Tnum1 → T2]	Tnum1 = number, Tnum4 = number,
Tf = number	T1 = [number * [number *number → number] → T0] }
Tx = [number *number → number]	
T0 = T2	

Equation	**Substitution**
Tf = number	{ T+ = [number *number → number],
Tx = [number *number → number]	Tnum1 = number, Tnum4 = number,
T0 = T2	T1 = [number * [number *number → number] → T0],
	Tf = [Tx * number → T2] }

(Tf = number) ∘ substitution = (number = [Tx * number → T2]) which is a wrong statement. **There is no solution to this equations system because the constraint cannot be solved.**

Part 2: Async Fun with TypeScript

2.2 (b)

The ‘Promise<R>’ allows us to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods (asyncMemo) return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

Part 3: Type Inference System

3.1

Typing rule define:

For every: type environment _Tenv,
variable _x1
expressions _e1 and
type expressions _S1, _U1:
If _Tenv o {_x1: _S1} |- _e1: _S1,
Then _Tenv |- (define _x1 _e1) : void

Typing rule set!:

For every: type environment _Tenv,
variable _x1
expressions _e1 and
type expressions _S1, _U1:
If _Tenv |- _x1: _S1
If _Tenv |- _e1: _S1
Then _Tenv |- (set! _x1 _e1) : void