**PPL – Assignment4 – Part1**

**By Yair Derry & Gal Noy**

(a) The typing statement (f (g a)) : T2 is **false** because function f expects an input of type T1, not T2 as provided by function g. Therefore, the resulting type of the expression cannot be T2.

(b) The typing statement (lambda (x) (f x 100)) : [T2 → T3] is **false** because the type T2 is not necessarily restricted to a number. The expression (f x 100) implies that the second argument must be of type T2, but it does not specify that T2 is specifically a numeric type.

(c) The typing statement ((lambda (x) (f x))) : [T1 → T2] is **true** because the lambda function takes an argument x, and when applied to an input of type T1, it returns a value of type T2. The type of the bound variable x can be inferred through type inference, and it does not require any assumptions in the type environment on the left side.

(d) The typing statement (lambda (x) (f x y)) : [T1 → T3] is **true** because the lambda function accepts an argument x of type T1 and returns a value of type T3, assuming the left-hand side typing context. The additional variable y with type T2 does not affect the overall type of the lambda function.

1. ((lambda (f x1) (if x1 (f 1 x1) (f 3 x1))) + #t)

**Rename bound variables**:

((lambda (f x) (if x (f 1 x) (f 3 x))) + #t)

**Assign type variables to all sub-expressions**:

T0 = ((lambda (f x) (if x1 (f 1 x) (f 3 x))) + #t)

T1 = (lambda (f x) (if x (f 1 x) (f 3 x)))

T2 = (if x (f 1 x) (f 3 x))

T3 = (f 1 x), T4 = (f 3 x), T5 = f, T6 = x, T7 = +, T8 = #t, T9 = 1, T10 = 3

**Construct type equations**:

T1 = [T7 \* T8 -> T0]

T1 = [T5 \* T6 -> T2]

T5 = [T9 \* T6 -> T3]

T5 = [T10 \* T6 -> T4]

T2 = T3 = T4

T8 = Boolean

T9 = T10 = number

T7 = [number \* number -> number]

T7 = T5

T8 = T6.

**Solve the equations**:

T5 = [T9 \* T6 -> T3]

T7 = [number \* number -> number]

T6 = T8 = Boolean

* T5 = [T9 \* boolean -> T3]

T5 = T7

* T7 = [number \* number -> number]
* [T9 \* boolean -> T3] = [number \* number -> number]

Contradiction!

1. ((lambda (f1 x1 y1) (f1 x1 y1)) \* 1 3)

**Rename bound variables**:

((lambda (f x y) (f x y)) \* 1 3)

**Assign type variables to all sub-expressions**:

T0 = ((lambda (f x y) (f x y)) \* 1 3)

T1 = (lambda (f x y) (f x y))

T2 = (f x y)

T3 = f, T4 = x, T5 = y, T6 = \*, T7 = 1, T8 = 3

**Construct type equations**:

T1 = [T6 \* T7 \* T8 -> T0]

T1 = [T3 \* T4 \* T5 -> T2]

T3 = [T4 \* T5 -> T2]

T6 = [number \* number -> number]

T7 = T8 = number

**Solve the equations**:

T1 = [T6 \* T7 \* T8 -> T0]

T1 = [T3 \* T4 \* T5 -> T2]

* T3 = T6, T7 = T4, T8 = T5, T0 = T2

T7 = T8 = number

* T4 = T5 = number

T3 = [T4 \* T5 -> T2]

T6 = [number \* number -> number]

T3 = T6

* T3 = [number \* number -> number]
* T2 = number

T0 = T2

* T0 = number

There are no contradictions, so we infer that the type of the expression is a number.

(b) The typing statement (lambda (x) (f x 100)) : [T2 → T3] is **false** because the type T2 is not necessarily restricted to a number. The expression (f x 100) implies that the second argument must be of type T2, but it does not specify that T2 is specifically a numeric type.

**Question 3**:

// Purpose: compute the type of a define

// Typing rule:

//   (define (var : texp) val)

//   if type<var>(tenv) = t

//      type<val>(tenv) = t

// then type<(define (var : texp) val)>(tenv) = void