## Part 1: Theoretical Questions

- 1. (a) False g returns T2 type, but f gets an argument of T1 type
  - (b) False x is T1, so the correct answer is T1->T3
  - (c) True
  - (d) True

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

## STAGE I+II:

Exp	Var
((lambda (f x1) (if x1 (f 1 x1) (f 3 x1))) + #t):	T0
(lambda (f x1) (if x1 (f 1 x1) (f 3 x1))):	T1
(if x1 (f 1 x1) (f 3 x1)):	Tif
x1:	Tx1
(f 1 x1):	Tthen
1:	Tnum1
(f 3 x1):	Talt
3:	Tnum3
+:	T+
#t	T#t

### STAGE III:

EXP	Eq
((lambda (f x1) (if x1 (f 1 x1) (f 3 x1))) + #t):	T1 = [T+ * T#t] -> T0
(lambda (f x1) (if x1 (f 1 x1) (f 3 x1))):	T1 = [Tf * Tx1] -> Tif
(if x1 (f 1 x1) (f 3 x1)):	Tif = Tx1 -> Tthen,
	Tx1 = boolean,
	Tthen = Talt
(f 1 x1):	Tf = [Tnum1 * Tx1] -> Tthen
1:	Tnum1 = number
(f 3 x1):	Tf = [Tnum3 * Tx1] -> Talt

3: Tnum3 = number

+: T+ = [number \* number] -> number

#t: T#t = boolean

# STAGE IV:

Eq
T1 = [T+ * T#t] -> T0
T1 = [Tf * Tx1] -> Tif
Tif = Tx1 -> Tthen
Tx1 = boolean
Tthen = Talt
Tf = [Tnum1 * Tx1] -> Tthen
Tnum1 = number
Tf = [Tnum3 * Tx1] -> Talt
Tnum3 = number
T+ = [number * number] -> number
T#t = boolean

Eq	Sub
T1 = [T+ * T#t] -> T0	T1 = [T+ * T#t] -> T0
T1 = [Tf * Tx1] -> Tif	
Tif = Tx1 -> Tthen	
Tx1 = boolean	
Tthen = Talt	
Tf = [Tnum1 * Tx1] -> Tthen	
Tnum1 = number	
Tf = [Tnum3 * Tx1] -> Talt	
Tnum3 = number	
T+ = [number * number] -> number	
T#t = boolean	

Eq	Sub
T1 = [Tf * Tx1] -> Tif	T1 = [T+ * T#t] -> T0
Tif = Tx1 -> Tthen	
Tx1 = boolean	
Tthen = Talt	
Tf = [Tnum1 * Tx1] -> Tthen	
Tnum1 = number	
Tf = [Tnum3 * Tx1] -> Talt	
Tnum3 = number	
T+ = [number * number] -> number	
T#t = boolean	

Eq	Sub
Tif = Tx1 -> Tthen	T1 = [T+ * T#t] -> T0
Tx1 = boolean	
Tthen = Talt	
Tf = [Tnum1 * Tx1] -> Tthen	
Tnum1 = number	
Tf = [Tnum3 * Tx1] -> Talt	
Tnum3 = number	
T+ = [number * number] -> number	
T#t = boolean	
T+ = Tf	
Tx1 = T#t	
T0= Tif	

Eq	Sub
Tx1 = boolean	T1 = [T+ * T#t] -> T0
Tthen = Talt	Tif = Tx1 -> Tthen
Tf = [Tnum1 * Tx1] -> Tthen	
Tnum1 = number	
Tf = [Tnum3 * Tx1] -> Talt	
Tnum3 = number	
T+ = [number * number] -> number	
T#t = boolean	
T+ = Tf	
Tx1 = T#t	
T0= Tif	

Eq	Sub
Tthen = Talt	T1 = [T+ * T#t] -> T0
Tf = [Tnum1 * Tx1] -> Tthen	Tif = boolean -> Tthen
Tnum1 = number	Tx1 = boolean
Tf = [Tnum3 * Tx1] -> Talt	
Tnum3 = number	
T+ = [number * number] -> number	
T#t = boolean	
T+ = Tf	
Tx1 = T#t	
TO= Tif	

Eq	Sub
Tf = [Tnum1 * Tx1] -> Tthen	T1 = [T+ * T#t] -> T0
Tnum1 = number	Tif = boolean -> Talt
Tf = [Tnum3 * Tx1] -> Talt	Tx1 = boolean
Tnum3 = number	Tthen = Talt
T+ = [number * number] -> number	
T#t = boolean	
T+ = Tf	
Tx1 = T#t	
T0= Tif	

Eq	Sub
Tnum1 = number	T1 = [T+ * T#t] -> T0
Tf = [Tnum3 * Tx1] -> Talt	Tif = boolean -> Talt
Tnum3 = number	Tx1 = boolean
T+ = [number * number] -> number	Tthen = Talt
T#t = boolean	Tf = [number * boolean] -> Talt
T+ = Tf	
Tx1 = T#t	
T0= Tif	

Eq	Sub
Tf = [Tnum3 * Tx1] -> Talt	T1 = [T+ * T#t] -> T0
Tnum3 = number	Tif = boolean -> Talt
T+ = [number * number] -> number	Tx1 = boolean
T#t = boolean	Tthen = Talt
T+ = Tf	Tf = [number * boolean] -> Talt Tnum1 = number
Tx1 = T#t	- mani - maniser
T0= Tif	

Eq	Sub
Tnum3 = number	T1 = [T+ * T#t] -> T0
T+ = [number * number] -> number	Tif = boolean -> Talt
T#t = boolean	Tx1 = boolean
T+ = Tf	Tthen = Talt
Tx1 = T#t	Tf = [number * boolean] -> Talt Tnum1 = number
T= Tif	Than - hamber

Eq	Sub
T+ = [number * number] -> number	T1 = [T+ * T#t] -> T0
T#t = boolean	Tif = boolean -> Talt
T+ = Tf	Tx1 = boolean
Tx1 = T#t	Tthen = Talt
	Tf = [number * boolean] -> Talt
T0= Tif	Tnum1 = number
	Tnum3 = number

Eq	Sub
T#t = boolean	T1 = [[[number * number] -> number ] * T#t] -> T0
T+ = Tf	Tif = boolean -> Talt
Tx1 = T#t	Tx1 = boolean
T0= Tif	Tthen = Talt
10-111	Tf = [number * boolean] -> Talt
	Tnum1 = number
	Tnum3 = number
	T+ = [number * number] -> number

Eq	Sub
T+ = Tf	T1 = [[[number * number] -> number ] * boolean] -> T0
Tx1 = T#t	Tif = boolean -> Talt
T0= Tif	Tx1 = boolean
	Tthen = Talt
	Tf = [number * boolean] -> Talt
	Tnum1 = number
	Tnum3 = number
	T+ = [number * number] -> number
	T#t = boolean

Eq	Sub
Tx1 = T#t	T1 = [[[number * number] -> number ] * boolean] -> T0
T0= Tif	Tif = boolean -> Talt
	Tx1 = boolean
	Tthen = Talt
	Tf = [number * boolean] -> Talt
	Tnum1 = number
	Tnum3 = number
	T+ = [number * number] -> number
	T#t = boolean
	T+ = Tf

<u>Note:</u> for conclusion, we found out that T+ = Tf, which means number = boolean - **contradiction!** Therefore, there is no solution for this expression.

## (b) ((lambda (f1 x1 y1) (f1 x1 y1)) \* 1 3)

### STAGE I+II:

Ехр	Var
((lambda (f1 x1 y1) (f1 x1 y1)) * 1 3) :	T0
(lambda (f1 x1 y1) (f1 x1 y1) :	T1
(f1 x1 y1):	T2

f1: Tf1
x1: Tx1
y1: Ty1
\*: T\*

1: Tnum1

3: Tnum3

## STAGE III:

Exp Eq

((lambda (f1 x1 y1) (f1 x1 y1)) \* 1 3):  $T1 = [T* * Tnum1 * Tnum3] \rightarrow T0$ 

(lambda (f1 x1 y1) (f1 x1 y1): T1 = [Tf1 \* Tx1 \* Ty1] -> T2

(f1 x1 y1): Tf = [Tx1 \* Ty1] -> T2

\*: T\* = number \* number -> number

1: Tnum1 = number

3: Tnum3 = number

### STAGE IV:

Eq	Sub
T1 = [T* * Tnum1 * Tnum3] -> T0	
T1 = [Tf1 * Tx1 * Ty1] -> T2	
Tf = [Tx1 * Ty1] -> T2	
T* = number * number -> number	
Tnum1 = number	
Tnum3 = number	

Eq	Sub
T1 = [Tf1 * Tx1 * Ty1] -> T2	T1 = [T* * Tnum1
Tf = [Tx1 * Ty1] -> T2	
T* = number * number -> number	
Tnum1 = number	
Tnum3 = number	

Eq	Sub
Tf = [Tx1 * Ty1] -> T2	T1 = [T* * Tnum1
T* = number * number -> number	
Tnum1 = number	
Tnum3 = number	
T* = Tf1	
Tnum1 = Tx1	
Tnum3 = Ty1	
T0 = T2	

Eq	Sub
T* = number * number -> number	T1 = [T* * Tnum1* Tnum3] -> T0
Tnum1 = number	Tf1 = [Tx1 * Ty1] -> T2
Tnum3 = number	
T* = Tf1	
Tnum1 = Tx1	
Tnum3 = Ty1	
T0 = T2	

Eq	Sub
Tnum1 = number	T1 = [[number * number -> number] * Tnum1* Tnum3] -> T0
Tnum3 = number	Tf1 = [Tx1 * Ty1] -> T2
T* = Tf1	T* = number * number -> number
Tnum1 = Tx1	
Tnum3 = Ty1	
T0 = T2	

Eq	Sub
Tnum3 = number	T1 = [[number * number -> number] * number * Tnum3] -> T0
T* = Tf1	Tf1 = [Tx1 * Ty1] -> T2
Tnum1 = Tx1	T* = number * number -> number
Tnum3 = Ty1	Tnum1 = number
T0 = T2	

Eq	Sub
T* = Tf1	T1 = [[number * number -> number] * number * number ] -> T0
Tnum1 = Tx1	Tf1 = [Tx1 * Ty1] -> T2
Tnum3 = Ty1	T* = number * number -> number
T0 = T2	Tnum1 = number
10 - 12	Tnum3 = number

Eq	Sub
Tnum1 = Tx1	T1 = [[number * number -> number] * number * number] -> T0
Tnum3 = Ty1	T* = number * number -> number  Tf1 = [Tx1 * Ty1] -> T2  Tnum1 = number  Tnum3 = number
T0 = T2	
Tx1 = number	
Ty1 = number	
T2 = number	

Eq	Sub
Tnum3 = Ty1	T1 = [[number * number -> number] * number * number] -> T0
T0 = T2	T* = number * number -> number  Tf1 = [Tx1 * Ty1] -> T2  Tnum1 = number  Tnum3 = number
Tx1 = number	
Ty1 = number	
T2 = number	

Eq	Sub
T0 = T2	T1 = [[number * number -> number] * number * number] -> T0

Tx1 = number	Tf1 = [Tx1 * Ty1] -> T2
Ty1 = number	T* = number * number -> number
T2 = number	Tnum1 = number Tnum3 = number

Eq	Sub
Tx1 = number	T1 = [[number * number -> number] * number * number ] -> T2
Ty1 = number	Tf1 = [Tx1 * Ty1] -> T2
T2 = number	T* = number * number -> number Tnum1 = number Tnum3 = number T0 = T2

Eq	Sub
Ty1 = number	T1 = [[number * number -> number] * number * number ] -> T2
T2 = number	Tf1 = [number * Ty1] -> T2
	T* = number * number -> number
	Tnum1 = number
	Tnum3 = number
	T0 = T2
	Tx1 = number

Eq	Sub
T2 = number	T1 = [[number * number -> number] * number * number] -> T2
	Tf1 = [number * number] -> T2
	T* = number * number -> number
	Tnum1 = number
	Tnum3 = number
	T0 = T2
	Tx1 = number
	Ty1 = number

Eq	Sub
	T1 = [[number * number -> number] * number * number ] -> number
	Tf1 = [number * number] -> number
	T* = number * number -> number
	Tnum1 = number
	Tnum3 = number
	T0 = number
	Tx1 = number
	Ty1 = number
	T2 = number

#### (b)

#### Step 1

((lambda (f1 x1 y1) (f1 x1 y1)) \* 1 3) : T1 (lambda (f1 x1 y1) (f1 x1 y1)) : T2 (f1 x1 y1) : T3 x1 : T4 y1 : T5 f1 : T6 \* : T7 1 : T8 3 : T9

#### Step2

T1 = (T7 X T8 X T9) -> T3 T2 = (T6 X T4 X T5) -> T3 T6 = (T4 X T5) -> T3 T7 = (number X number) -> number T8 = T9 = number

#### Step 3

T1 = (((number X number) -> number) X number X number) -> T3
 T2 = (((T4 X T5) -> T3) X T4 X T5) -> T3
 T6 = T7
 T4 = T8 = number
 T5 = T9 = number

T1 = (((number X number) -> number) X number X number) -> T3
 T2 = (((T4 X T5) -> T3) X T4 X T5) -> T3

```
T6 = T7
T3 = (number X number) -> number)
T4 = T8 = number
T5 = T9 = number

• T2 = (((number X number) -> number) number X number) -> number
```

### Question 3.1

```
Typing rule:
```

```
// if type<val> = T1
// then type((define (var : texp) val)) (extend-tenv(var = T1, tenv)) = void
```