Ppl H.W 4 Answers:

Id1: 206226581

Id2: 322217472

Part 1:

Question 1

((lambda (x1 y1) (if (> x1 y1) #t #f)) 8 3)

Step 1 – renaming

((lambda (x y) (if (> x y) #t #f)) 8 3)

Step 2 - type variables

((lambda (x y) (if (> x y) #t #f)) 8 3)	T_0
(lambda (x y) (if (> x y) #t #f))	T_1
(if (> x y) #t #f)	T_2
(> x y)	T_3
>	$T_>$
x	T_X
Y	T_y
#t	$T_{\#t}$
#f	$T_{\#f}$
8	T_{num8}
3	T_{num3}

Step 3 - Construct type equations

((lambda (x y) (if (> x y) #t #f)) 8 3)	$T_1 = [T_{num8} \, X \, T_{num3} \, -> T_0]$
(lambda (x y) (if (> x y) #t #f))	$T1 = [T_x \, X \, T_y \, -> T_2]$
(if (> x y) #t #f)	$T2 = T_{\#t}$ and $T_{\#f} = T_{\#f}$
(> x y)	$T> = [T_X * T_y \, -> T_3]$
>	$T_> = [Number*Number \, -> Boolean]$
#t	$T_{\#t} = Boolean$
#f	$T_{\#f} = Boolean$
8	$T_{num8} = Number$
3	$T_{num3 = Number}$

Step 4 - Solve the equations

equations	Substitution
1. T1 = [Tnum8 X Tnum3 -> T0] 2. T1 = [Tx X Ty -> T2] 3. T2 = T#f 4. T#f = T#t 5. T> = [TX * Ty -> T3] 6. T> = [Number*Number -> Boolean] 7. T#f = Boolean 8. T#t = Boolean 9. Tnum8 = Number 10. Tnum3 = Number	{}
equations 2. T1 = [Tx X Ty -> T2] 3. T2 = T#f 4. T#f = T#t 5. T> = [TX * Ty -> T3] 6. T> = [Number*Number -> Boolean] 7. T#f = Boolean 8. T#t =Boolean 9. Tnum8 = Number 10. Tnum3 = Number	Substitution { T1 := [Tnum8 X Tnum3 -> T0] }
equations ~~2. T1 = [Tx X Ty -> T2]~~ 3. T2 = T#f 4. T#f = T#t 5. T> = [TX * Ty -> T3] 6. T> = [Number*Number -> Boolean] 7. T#f = Boolean 8. T#t = Boolean 9. Tnum8 = Number 10. Tnum3 = Number 11. Tx = Tnum8 12. Ty = Tnum3	Substitution { T1 := [Tnum8 X Tnum3 -> T0] }

13. T2 = T0	
equations 5. T> = [TX * Ty -> T3] 6. T> = [Number*Number -> Boolean] 7. T#f = Boolean 8. T#t = Boolean 9. Tnum8 = Number 10. Tnum3 = Number 11. Tx = Tnum8 12. Ty = Tnum3 13. T2 = T0	Substitution { T1 := [Tnum8 X Tnum3 -> T0] T2 := T#f T#f := T#t }
equations 6. T> = [Number*Number -> Boolean] 7. T#f = Boolean 8. T#t = Boolean 9. Tnum8 = Number 10. Tnum3 = Number 11. Tx = Tnum8 12. Ty = Tnum3 13. T2 = T0	Substitution { T1 := [Tnum8 X Tnum3 -> T0] T2 := T#f T#f := T#t T> := [TX * Ty -> T3] }
equations ~~6. T> = [Number*Number -> Boolean]~~ 7. T#f = Boolean 8. T#t = Boolean 9. Tnum8 = Number 10. Tnum3 = Number 11. Tx = Tnum8 12. Ty = Tnum3 13. T2 = T0 14. Tx =Number 15. Ty = Number 16. T3 = Boolean	Substitution { T1 := [Tnum8 X Tnum3 -> T0] T2 := T#f T#f := T#t T> := [TX * Ty -> T3] }

equations	Substitution
9. Tnum8 = Number 10. Tnum3 = Number 11. Tx = Tnum8 12. Ty = Tnum3 13. T2 = T0 14. Tx =Number 15. Ty = Number 16. T3 = Boolean	{ T1 := [Tnum8 X Tnum3 -> T0] T2 := Boolean T#f = Boolean T> := [TX * Ty -> T3] T#t = Boolean }
equations 11. Tx = Tnum8 12. Ty = Tnum3 13. T2 = T0 14. Tx =Number 15. Ty = Number 16. T3 = Boolean	Substitution { T1 := [Number X Number -> T0] T2 := Boolean T#f = Boolean T> := [TX * Ty -> T3] T#t = Boolean Tnum8 = Number Tnum3 = Number }
equations 13. T2 = T0 14. Tx =Number 15. Ty = Number 16. T3 = Boolean	Substitution { T1 := [Number X Number -> T0] T2 := Boolean T#f = Boolean T> := [Number * Number -> T3] T#t = Boolean Tnum8 = Number Tnum3 = Number Tx = Number Ty = Number }
equations 13. T2 = T0 14. Tx =Number 15. Ty = Number 16. T3 = Boolean	Substitution { T1 := [Number X Number -> T0] T2 := Boolean T#f = Boolean T> := [Number * Number -> T3] T#t = Boolean Tnum8 = Number Tnum3 = Number Tx = Number Ty = Number }

equations	Substitution
16. T3 = Boolean	{ T1 := [Number X Number -> Boolean] T2 := Boolean T#f = Boolean T> := [Number * Number -> T3] T#t = Boolean Tnum8 = Number Tnum3 = Number Tx = Number Ty = Number T0 = Boolean }
equations	Substitution
	{ T1 := [Number X Number -> Boolean] T2 := Boolean T#f = Boolean T> := [Number * Number -> Boolean] T#t = Boolean Tnum8 = Number Tnum3 = Number Tx = Number Ty = Number T0 = Boolean T3 = Boolean }

Question 2

a) {f:[T1->T2], x: T1} |- (f x): T2

true : because f is from T1 to T2. when perform f on x (from type T1) the function returns a variable from type T2. Therefore the statement (f x) : T2 is valid.

b) {f:[T1->T2],g: [T2->T3]}, x: T2}|- (f g x): T3

false : the number of parameters passed to f is invalid (f is define to get one parameter and receives two).

c) {f:[T2->T1],g: [T1->T2], x: T1}|- (f (g x)): T1

true : when perform g on x , the function returns variable of type T2 (because g is from T1 to T2) and the statement (g x):T2 is valid. F is from T2 to T1 and (g x) is from type T2. Therefore (f (g x)) return T1 and the statement (f (g x)) is valid.

d) {f:[T2->Number],, x: Number}|- (f x x): Number

false : the number of parameters passed to f is invalid (f is define to get one parameter and receives two).

Question 3

cons: T1 * T2 -> (Pair T1 T2)
car: (Pair T1 T2) -> T1
cdr: (Pair T1 T2) -> T2

Question 4

(Define f (lambda (x:T) : [T -> T* T* T] (values x x x)))

Question 5

a) **T1 , T2**
MGU : {T1 = T2}
b) **Number , Number**
MGU : {}
c) **[T1*[T1->T2]->Number] , [[T3->Number]*[T4->Number]->N]**
MGU : { T2 =Number , T1=[T3->Number]=T4}
d) **[T1->T1] , [T1->[Number->Number]**
MGU : {T1 = Number->Number}

Part 2:

**Regarding this part we implemented "values" as PrimOp and we didn't added support to nested tuples.

Answers 2.3:

```
(define f: [Number ->[Number*Number]
        (lambda (x: number)
                (values x (+ x 1))))

(define g: [T1-> [string*T1]]
        (lambda (x: T1)
                (values “x” x)))
```

Part 4:

Answers to Q.4 - 1.b:

Benefits of "promise" over "callback" :

1. Using promises produce clearer and cleaner code than using callbacks.
2. Promise has a version of "try – catch" mechanism to allow smoothly continuation of the code.
3. The type of a function that return promise is clearer than function that uses callbacks instead (usually seems messy and not understandable type).
4. Chaining results of functions using promise is simpler than functions that use callbacks.