PPL - Assignment 4

ID : 318425337

Part 1: Theoretical questions:

Q1: Perform typing inference for the expression:

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

Stage I: Rename bound variables:

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

Stage II: Assign type variables for every sub expression:

Expression	Variable
((lambda (x y) (if (> x y) #t #f)) 8 3)	T0
(lambda (x y) (if (> x y) #t #f))	T1
(if (> x y) #t #f)	T2
(> x y)	T3
>	T>
X	Tx
Y	Ty
#t	$T_{\#t}$
#f	$T_{\#f}$
8	Tnum8
3	Tnum3

Stage III: Construct type equations. The equations for the sub-expressions are:

Expression	Equation
((lambda (x y) (if (> x y) #t #f)) 8 3)	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$
(lambda (x y) (if (> x y) #t #f))	$T_1 = [T_x * T_y \rightarrow T_2]$
(if (> x y) #t #f)	$T_2 = T_{\#t}\ and\ T_{\#t} = T_{\#f}$
(> x y)	$T_> = [Tx * Ty \rightarrow T3]$

The equations for the primitives are:

Expression	Equation
`>`	$T_{>} = [number * number \rightarrow number]$
`#t`	$T_{\#t} = boolean$
`#f`	$T_{\#f} = boolean$
`8`	$T_{num8} = number$
`3`	$T_{num3} = number$

Stage IV: Solve the equations.

Step 1:

Equation	**Substitution**
$\mathbf{T1 = [Tnum8 * Tnum3 \rightarrow T0]}$	**T1 = [Tnum8 * Tnum3 → T0**
$T1 = [T_x * T_y \rightarrow T2]$	
$T2 = T_{\#t}$	
T_{test} = boolean	
$T_{\#t} = T_{\#f}$	
$T_{>} = [T_x * T_y \rightarrow T_{test}]$	
$T_{>}$ = [number * number → boolean]	
$T_{\#t}$ = boolean	
$T_{\#f}$ = boolean	
T_{num8} = number	
T_{num3} = number	

Step 2:

Equation	Substitution
$T_1 = [T_x * T_y \rightarrow T_{if}]$	T1 = [Tnum8 * Tnum3 → T0
$T_{if} = T_{\#t}$	
$T_{test} = \text{boolean}$	
$T_{\#t} = T_{\#f}$	
$T_> = [T_x * T_y \rightarrow T_{test}]$	
$T_> = [\text{number} * \text{number} \rightarrow \text{boolean}]$	
$T_{\#t} = \text{boolean}$	
$T_{\#f} = \text{boolean}$	
$T_{num8} = \text{number}$	
$T_{num3} = \text{number}$	

Step 3:

Equation	Substitution
$T_{if} = T_{\#t}$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$
$T_{test} = \text{boolean}$	
$T_{\#t} = T_{\#f}$	
$T_> = [T_x * T_y \rightarrow T_{test}]$	
$T_> = [\text{number} * \text{number} \rightarrow \text{boolean}]$	
$T_{\#t} = \text{boolean}$	
$T_{\#f} = \text{boolean}$	
$T_{num8} = \text{number}$	
$T_{num3} = \text{number}$	
$T_{x1} = T_{num8}$	
$T_{y1} = T_{num3}$	
$T_{if} = T_0$	

Step 4:

Equation	Substitution
$T_{\#t} = T_{\#f}$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ $T_{if} = T_{\#t}$ $T_{test} =$ boolean
$T_> = [T_{x1} * T_{y1} \rightarrow T_{test}]$	
$T_> =$ [number * number → boolean]	
$T_{\#t} =$ boolean	
$T_{\#f} =$ boolean	
$T_{num8} =$ number	
$T_{num3} =$ number	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_{if} = T_0$	

Step 5 :

Equation	Substitution
$T_> = [T_{x1} * T_{y1} \rightarrow$ boolean]	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ $T_{if} = T_{\#f}$ $T_{test} =$ boolean $T_{\#t} = T_{\#f}$
$T_> =$ [number * number → boolean]	
$T_{\#t} =$ boolean	
$T_{\#f} =$ boolean	
$T_{num8} =$ number	
$T_{num3} =$ number	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_{if} = T_0$	

Step 6 :

Equation	Substitution
$T_{>}$ = [number * number → boolean]	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ $T_{if} = T_{\#f}$ T_{test} = boolean $T_{\#t} = T_{\#f}$ $T_{>} = [T_{x1} * T_{y1} \rightarrow$ boolean]
$T_{\#t}$ = boolean	
$T_{\#f}$ = boolean	
T_{num8} = number	
T_{num3} = number	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_{if} = T_0$	

Step 7 :

Equation	Substitution
$T_{\#t}$ = boolean	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ $T_{if} = T_{\#f}$ T_{test} = boolean $T_{\#t} = T_{\#f}$ $T_{>} = [T_x * T_y \rightarrow$ boolean] $T_{>}$ = [number * number → boolean]
$T_{\#f}$ = boolean	
T_{num8} = number	
T_{num3} = number	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_{if} = T_0$	
T_x = number	
T_y = number	
boolean = boolean	

Step 8 :

Equation	Substitution
$T_{\#f}$ = boolean	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ $T_{if} = T_{\#f}$ T_{test} = boolean $T_{\#t} = T_{\#f}$ $T_> = [T_x * T_y \rightarrow$ boolean$]$ $T_> =$ [number * number → boolean] $T_{\#t}$ = boolean
T_{num8} = number	
T_{num3} = number	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_{if} = T_0$	
T_x = number	
T_y = number	
boolean = boolean	

Step 9 :

Equation	Substitution
T_{num8} = number	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_> = [T_x * T_y \rightarrow$ boolean$]$ $T_> =$ [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean
T_{num3} = number	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_{if} = T_0$	
T_x = number	
T_y = number	
boolean = boolean	

Step 10 :

Equation	Substitution
$T_x = T_{num8}$	$T_1 =$ [number * number → T_0] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_> = [T_{x1} * T_{y1} \rightarrow$ boolean$]$ $T_> =$ [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number
$T_y = T_{num3}$	
$T_{if} = T_0$	
T_x = number	
T_y = number	
boolean = boolean	

Step 11 :

Equation	Substitution
T_x = number	T_1 = [number * number → T_0] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_>$ = [T_{x1} * T_{y1} → boolean] $T_>$ = [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number
T_y = T_{num3}	
T_{if} = T_0	
T_x = number	
T_y = number	
boolean = boolean	

Step 12 :

Equation	Substitution
T_y = T_{num3}	T_1 = [number * number → T_0] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_>$ = [number * T_{y1} → boolean] $T_>$ = [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number T_x = number
T_{if} = T_0	
T_x = number	
T_y = number	
boolean = boolean	

Step 13 :

Equation	Substitution
T_{y1} = number	T_1 = [number * number → T_0] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_>$ = [number * T_{y1} → boolean] $T_>$ = [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number T_x = number
T_{if} = T_0	
T_{x1} = number	
T_{y1} = number	
boolean = boolean	

Step 14 :

Equation	Substitution
$T_{if} = T_0$	T_1 = [number * number → T_0] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_>$ = [number * number → boolean] $T_>$ = [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number T_x = number T_y = number
T_{x1} = number	
T_{y1} = number	
boolean = boolean	

Step 15 :

Equation	Substitution
T_{x1} = number	T_1 = [number * number → boolean] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_>$ = [number * number → boolean] $T_{\#t}$ = boolean $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number T_{x1} = number T_{y1} = number T_0 = boolean
T_{y1} = number	
boolean = boolean	

😊 😊 😊 **Step 16** 😊 😊 😊**:**

Equation	Substitution
	T_1 = [number * number → boolean] T_{if} = boolean T_{test} = boolean $T_{\#t}$ = boolean $T_>$ = [number * number → boolean] $T_{\#f}$ = boolean T_{num8} = number T_{num3} = number T_x = number T_y = number

	T_0 = boolean

Q2: Are these typing statements true? Explain.

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

true :

given f:[T1->T2] , we can see that $T_{f=}$ [T1 -> T2], because the type of a function is it's return type , and so it is T2

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

false :

the types are incompatible, **f:[T1->T2] , f gets 1 parameter of type T1 , and we call f on 2 arguments** ☹

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

true.

g: [T1->T2] and x : T1 makes g return a T2 without error of the type , the n f gets a valid argument from g (T2) and returns T1 ,which is the return type.

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

false

f gets 1 parameter of type T2 , but it was called with 2 parameters .

Q3: What is the type of the following primitive operators:

a. cons :

[x: T1, y: T2) --> Pair(T1, T2]

Cons receives two variables , x of type T1 , y of type T2 ,as parameters and returns a pair which has the values inside it (x as first one , and y as the second)

b. car

[Pair (T1, T2)) --> T1]

Car receives a tuple as parameter , and returns the first variable of the parameter in tuple , which type is T1

c. cdr

[Pair (T1, T2)) --> T2]

Cdr receives a tuple as parameter , and returns the second variable of the parameter in tuple , which type is T2

Q4: Write the type of the following function: (Define f (lambda (x) (values x x x)))

[T1 -> (T1 * T1 * T1)]

Q5. Write the MGU of the following expressions, or state that there is no such MGU.

a. T1 , T2 =>

MGU is {T1 = T2} – since this unifier does a substitution only for the critical types

b. Number , Number =>

MGU is { } (empty unifier) - because numbers stays numbers 😊

c. [T1*[T1->T2]->Number] , [[T3->Number]*[T4->Number]->N]

MGU is **{T1=[T3->Number], T4=[T3->Number], T2=Number}**

d. [T1->T1] , [T1->[Number->Number]

MGU is {T1 = [Number->Number] }

PART 2 .3:

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

(define g: [T1 -> Tuple (string , T1)]
        (lambda (x: T1): (string , T1)
                (values "x" x)))
```

PART 3 :

```
}



export function* braid(generator1: () => Generator, generator2: () => Generator) :Generator{
    const g1 = generator1();
    const g2 = generator2();

    let gens = [g1, g2];

    let nxt1 = g1.next();
    let nxt2 = g2.next();

    while(!nxt1.done || !nxt2.done){
        if(!nxt1.done){
            yield nxt1.value;
        }
        if(!nxt2.done){
            yield nxt2.value;
        }

        nxt1 = g1.next();
        nxt2 = g2.next();

    }
}
```

```
function* biased(g1: () => Generator, g2: () => Generator): Generator {
    let gen1 = g1();
    let gen2 = g2();
    let nxt1=gen1.next().value;
    let nxt2=gen2.next().value;

    while (nxt1!=undefined || nxt2!=undefined)
    {
        if (nxt1!=undefined)
        {
            yield nxt1
            nxt1=gen1.next().value
        }
        if (nxt1!=undefined)
        {
            yield nxt1
            nxt1=gen1.next().value
        }
        if (nxt2!=undefined)
        {
            yield nxt2
            nxt2=gen2.next().value
        }

    }
}
```

PART 4:

```
//Question 1
export function f(x: number): Promise<number> {
    return new Promise<number>((resolve : any, reject: any) => {
        if (x === 0) {
            reject("ERROR in function f : division by 0 !");
            return;
        }
            resolve(1 / x);
    });
}


export function g(x: number): Promise<number> {
    return new Promise<number>((resolve, reject) => {
            resolve(x * x);
            return;
    });
}


export function h(x: number): Promise<number> {
    return new Promise<number>((resolve, reject) => {
       g(x)
        .then((res) => f(res) )
        .then((res) => resolve(res) )
        .catch((error) => reject(error) );
    });
}
```

Code also in the next page , and Part 4 .b :

```
41
42  //Question2
43  export const slower = <T>(proms :[Promise<any>, Promise<any>]):Promise <any> => {
44      return new Promise(resolve => {
45        const wrap_promise = (index: number) => (value: T) => {
46          if(race.add(index).size === 2){
47              resolve(`(${index}, ${value})`)
48          }
49        }
50        const race=new Set
51        proms.map((prom,index)=>{
52        prom.then(wrap_promise(index))
53        })
54      })
55    }
56
```

Part 4: What are the benefits of the promise interface compared to the callback interface?

a) Functions in promise are very close to the synchronous interface (except that it returns result of Promise<T>) , while Callbacks are more complicated .
b) Error handling is much easier in the Promise interface , we can use catch and reject , while in the CallBack , handling error require more work (for each error).
c) We can use the .then() calls in sequential order , which help us build compound functions , and it assure that we have a synchronization .