**PPL 4**

1. 1.First step is renaming – x and y occur more then once.

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

2. Now we will assign the type variables for every sub expression

|  |  |
| --- | --- |
| Expression | Variable |
| ((lambda (x1 y1) (if (> x1 y1) #t #f)) 8 3) | T0 |
| (lambda (x1 y1) (if (> x1 y1) #t #f)) | T1 |
| (if (> x1 y1) #t #f) | T2 |
| (> x1 y1) | T3 |
| > | T> |
| X | Tx |
| Y | Ty |
| #t | T#t |
| #f | T#f |
| 8 | Tnum8 |
| 3 | Tnum3 |

3. We will construct and the equations :

|  |  |
| --- | --- |
| Expression | Equation |
| ((lambda (x) (if (> x1 y1) #t #f)) 8) | T1 = [Tnum8 \* Tnum3 -> T0] |
| (lambda (x) (if (> x1 y1) #t #f)) | T1 = [Tx\*Ty -> Tif] |
| (if (> x1 y1) #t #f) | Tif = T#t |
| (> x1 y1) | T> = [Tx \* Ty-> T2] |
| > | T> = [Number \* Number -> Boolean] |
| #t | T#t = Boolean |
| #f | T#f = Boolean |
| 8 | Tnum8 = Number |
| 3 | Tnum3 = Number |

4. Now we will solve the equation

Substitution {}

|  |
| --- |
| T1 = [Tnum8 \* Tnum3 -> T0] |
| T1 = [Tx \* Ty -> Tif] |
| Tif = T#t |
| T#f = T#t |
| T> = [Tx \* Ty -> T2] |
| T> = [Number \* Number -> Boolean] |
| T#t = Boolean |
| T#f = Bolean |
| Tnum8 = Number |
| TNum3 = Number |

Substitution { T1 = [Tnum8 \* Tnum3-> T0]}

|  |
| --- |
| T1 = [Tx \* Ty -> Tif] |
| Tif = T#t |
| T#t = T#f |
| T> = [Tx \* Ty -> T2] |
| T> = [Number \* Number -> Boolean] |
| T#t = Boolean |
| T#f = Boolean |
| Tnum8 = Number |
| TNum3 = Number |

Substitution : {T1 := [Tnum8 \* Tnum3 -> T0]}

|  |
| --- |
| Tif = T#t |
| T#t = T#f |
| T> = [Tx \* Ty -> T2] |
| T> = [Number \* Number -> Boolean] |
| T#t = Boolean |
| T#f = Boolean |
| Tnum8 = Nmber |
| TNum3 = Number |
| Tx = Tnum8 |
| Ty = Tnum3 |
| T0 = T2 |

Substitution : { T1 = [Tnum8 \* Tnum3-> T0] ,T#t = T#f = T2}

|  |
| --- |
| T3 = Boolean |
| T> = [Tnum8 \* Tnum3 -> T3] |
| T> = [Number \* Number -> Boolean] |
| T#t = Boolean |
| T#f = Boolean |
| Tnum8 = Number |
| Tnum3 = Number |
| Tx = Tnum8 |
| Ty = Tnum3 |
| T0 = T2 |

Substitution : { T1 = [Tnum8\* Tnum3-> T0] , T#f = T#t = T2, T3 = Boolean , T> = [Tnum8 \* Tnum3 -> Boolean]}

|  |
| --- |
| T> = [Number \* Number -> Boolean] |
| T#t = Boolean |
| T#f = Boolean |
| Tnum8 = Number |
| Tnum3 = Number |
| Tx = Tnum8 |
| Ty = Tnum3 |
| T0 = T2 |

Substitution: { T1 = [Tnum8 \* Tnum3-> T0] , T#f = T#t = T2, T3 = Boolean , T> = [Tnum8 \* Tnum3 -> Boolean]}

|  |
| --- |
| T#t = Boolean |
| T#f = Boolean |
| Tnum8 = Number |
| Tnum3 = Number |
| Tx = Tnum8 |
| Ty = Tnum3 |
| T0 = T2 |

Add all 4 top because it is the same and basic

Substitution: { T1 = [Tnum8\* Tnum3-> T0] , T#f = T#t = T2, T3 = Boolean , T> = [Tnum8 \* Tnum3 -> Boolean], T#t = Boolean, T#f = Boolean, Tnum8 = Number, Tnum3 = Number}}

|  |
| --- |
| Tx = Tnum8 |
| Ty = Tnum3 |
| T0 = T2 |

Substitution: { T1 = [Tnum8\* Tnum3-> T0] , T#f = T#t = T2, T3 = Boolean , T> = [Tnum8 \* Tnum3 -> Boolean], T#t = Boolean, T#f = Boolean, Tnum8 = Number, Tnum3 = Number

Tx = Number, Ty = Number, T0 = Boolean}

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

b. {f:[T1->T2],g: [T2->T3]}, x: T2}|- (f g x): T3 – Wrong – number of parameters for f is incorrect

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

d. {f:[T2->Number],, x: Number}|- (f x x): Number -- Wrong – number of parameters for f is incorrect

3) cons – [ T1 \* T2 -> Pair(T1, T2) ]

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

cdr – [ Pair(T1, T2) -> T2 ]

4) (Define f (lambda (x) (values x x x))) -> [ T1 -> (T1 \* T1 \* T1) ]

5)

a. MGU - {T1 = T2}

b. There are no restrictions so basically the MGU is {} – there are no restrictions.

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

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

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

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

***Part 2***

We implemented values as a prime op, we did not support nested tuples.

(define (f: [number -> (number\*number)])

(lambda ((x: Number)) : ( Number \* Number)

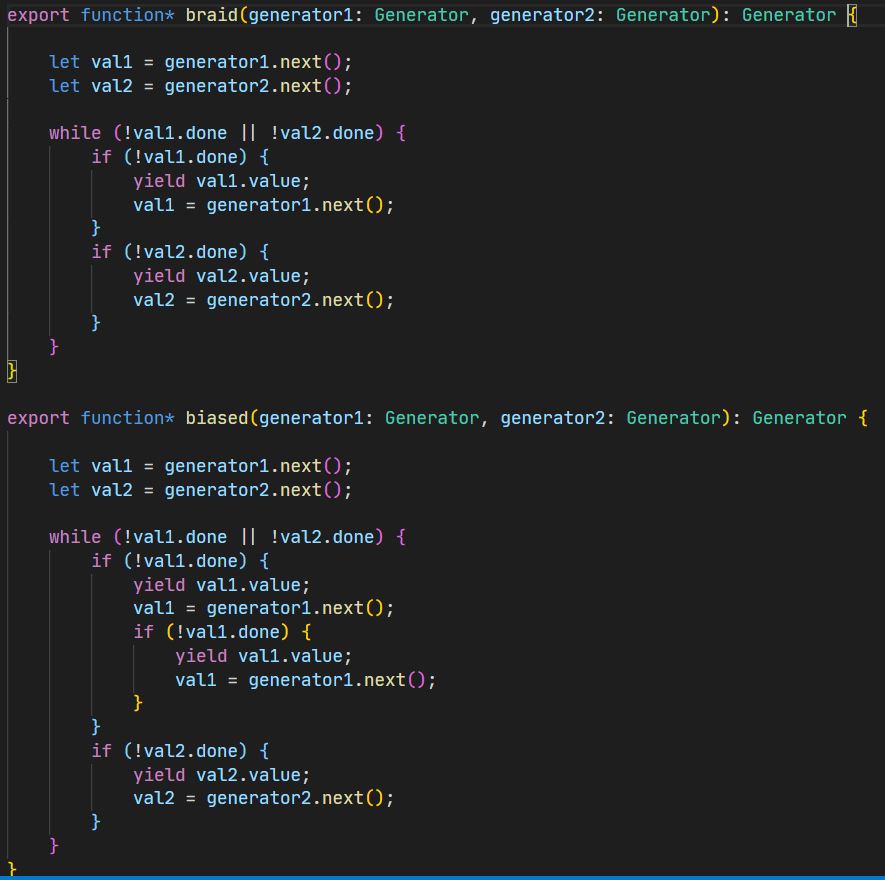
(values x (+ x 1))))

(define (g: [T -> (string \* T)])

(lambda ((x: T)) : ( String \* T)

(values “x” x)))

***Part 3***



***Part 4***

Promise is like result compared to callback, it makes error checking much simpler and lets the programmer focus on the ‘successful part’ independent from the error and it is most comfortable when there are a lot of places where the error could occur. It only requires from the developer to create one error handler in a chain of calls.

Promise allows us to chain results.

Promise signatures are much simpler than callback signatures.

