Part 1 q.1

a. false, g gets T1 argument but we supply to g the param a which its number.

b. false, f gets as parameter T1 but f is T2-> T1.

c: True, Empty -> T2.

d. True we can notice the f get T1, T2 arguments, lambda f gets x,y and y is type of T2. But x is not defined, so we can inherence that its type of T1, that’s why the body oh the lambda which gets T1’s arguments and calling f. we can notice that f returning T3 and that’s why the type of the function is T1->T3. because f: T1\*T2->T3, and y is T2. but if x is not T1 the function not work

Part 1 question 2 a

assign: ((lambda (f1 x11) (f1 1 x11)) + #t) T0

(lambda (f1 x11) (f1 1 x11)) T1

(f1 1 x11) T2

f1 Tf1

|  |  |
| --- | --- |
| Equation | Substitution |
| T1= T+\*Tboolean ->T0 |  |
| T1 = Tf1\*Tx11 -> T2 |  |
| Tf1= Tnum1\*Tx11 -> T2 |  |
| Tnum1= Number |  |
| T+= Number\*Number->Number |  |
| Tboolean=Boolean |  |

1 Tnum1

x11 Tx11

+ T+

#t Tboolean

|  |  |
| --- | --- |
| Equation | Substitution |
|  | T1= T+\*Tboolean ->T0 |
| T1 = Tf1\*Tx11 -> T2 |  |
| Tf1= Tnum1\*Tx11 -> T2 |  |
| Tnum1= Number |  |
| T+= Number\*Number->Number |  |
| Tboolean=Boolean |  |

[T+\*TBoolean -> T0] = [Tf1\*Tx11->T2]

|  |  |
| --- | --- |
| Equation | Substitution |
|  | T1= T+\*Tboolean ->T0 |
| Tnum1= Number | Tf1= Tnum1\*Tx11 -> T2 |
| T+= Number\*Number->Number |  |
| Tboolean=Boolean |  |
| Tf1=T+ |  |
| Tx11=Tboolean |  |
| T2=T0 |  |

|  |  |
| --- | --- |
| Equation | Substitution |
|  | T1= T+\*Tboolean ->T0 |
|  | Tf1= Number\*Tx11 -> T2 |
| T+= Number\*Number->Number | Tnum1= Number |
| Tboolean=Boolean |  |
| Tf1=T+ |  |
| Tx11=Tboolean |  |
| T2=T0 |  |

|  |  |
| --- | --- |
| Equation | Substitution |
|  | T1= [Number\*Number->Number ]\*Tboolean ->T0 |
|  | Tf1= Number\*Tx11 -> T2 |
|  | Tnum1= Number |
| Tboolean=Boolean | T+= Number\*Number->Number |
| Tf1=T+ |  |
| Tx11=Tboolean |  |
| T2=T0 |  |

|  |  |
| --- | --- |
|  | T1= [Number\*Number->Number ]\*Boolean ->T0 |
|  | Tf1= Number\*Tx11 -> T2 |
|  | Tnum1= Number |
|  | T+= Number\*Number->Number |
| Tf1=T+ | Tboolean=Boolean |
| Tx11=Tboolean |  |
| T2=T0 |  |

[Number\*Tx11 -> T2] =[ Number\*Number->Number]

|  |  |
| --- | --- |
|  | T1= [Number\*Number->Number ]\*Boolean ->T0 |
|  | Tf1= Number\*Number -> T2 |
| T2=T0 | Tnum1= Number |
| Tx11=Number | T+= Number\*Number->Number |
| T2=Number | Tboolean=Boolean |
|  | Tx11=Number |
|  |  |

|  |  |
| --- | --- |
|  | T1= [Number\*Number->Number ]\*Boolean ->T0 |
|  | Tf1= Number\*Boolean -> T0 |
|  | Tnum1= Number |
| Tx11=Number | T+= Number\*Number->Number |
| T2=Number | Tboolean=Boolean |
|  | Tx11=Boolean |
|  | T2=T0 |
|  |  |

Tx11= Number and Tx11= Boolean .

We get conflicting equation , so the expression is not well typed.

Part 1 question 2 b

((lambda (f1 x1) (f1 x1 1)) + \*) turns into ((lambda (f x) (f x 1)) + \*)

((lambda (f x) (f x 1)) + \*) T0

(lambda (f x) (f x 1)) T1

(f x 1) T2

f Tf

x Tx

1 Tnum1

+ T+

\* T\*

|  |  |
| --- | --- |
| Equation | Subtitiution |
| T1=T+ \* T\* -> T0 |  |
| T1= Tf \* Tx -> T2 |  |
| Tf=Tx \* Tnum1 -> T2 |  |
| Tnum1=Number |  |
| T+=Number\*Number->Number |  |
| T\*=Number\*Number->Number |  |

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=T+ \* T\* -> T0 |
| T1= Tf \* Tx -> T2 |  |
| Tf=Tx \* Tnum1 -> T2 |  |
| Tnum1=Number |  |
| T+=Number\*Number->Number |  |
| T\*=Number\*Number->Number |  |

[Tf\*Tx->T2]=[T+ \* T\* -> T0]

Tf=T+

Tx=T\*

T2=T0

|  |  |
| --- | --- |
| Equation | Subtitiution |
| Tf=Tx \* Tnum1 -> T2 | T1=T+ \* T\* -> T0 |
| Tnum1=Number |  |
| T+=Number\*Number->Number |  |
| T\*=Number\*Number->Number |  |
| Tf=T+ |  |
| Tx=T\* |  |
| T2=T0 |  |

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=T+ \* T\* -> T0 |
| Tnum1=Number | Tf=Tx \* Tnum1 -> T2 |
| T+=Number\*Number->Number |  |
| T\*=Number\*Number->Number |  |
| Tf=T+ |  |
| Tx=T\* |  |
| T2=T0 |  |

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=T+ \* T\* -> T0 |
|  | Tf=Tx \* Number -> T2 |
| T+=Number\*Number->Number | Tnum1=Number |
| T\*=Number\*Number->Number |  |
| Tf=T+ |  |
| Tx=T\* |  |
| T2=T0 |  |

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=[ Number\*Number->Number] \* T\* -> T0 |
|  | Tf=Tx \* Number -> T2 |
|  | Tnum1=Number |
| T\*=Number\*Number->Number | T+=Number\*Number->Number |
| Tf=T+ |  |
| Tx=T\* |  |
| T2=T0 |  |

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=[ Number\*Number->Number] \* [Number\*Number->Number] -> T0 |
|  | Tf=Tx \* Number -> T2 |
|  | Tnum1=Number |
|  | T+=Number\*Number->Number |
| Tf=T+ | T\*=Number\*Number->Number |
| Tx=T\* |  |
| T2=T0 |  |

[Tx \* Number -> T2] = [Number\*Number->Number]

Tx=Number

T2=Number

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=[ Number\*Number->Number] \* [Number\*Number->Number] -> T0 |
| T2=T0 | Tf=[ Number\*Number->Number] \* Number -> T2 |
| Tx=Number | Tnum1=Number |
| T2=Number | T+=Number\*Number->Number |
|  | T\*=Number\*Number->Number |
|  | Tx= Number\*Number->Number |
|  |  |

|  |  |
| --- | --- |
| Equation | Subtitiution |
|  | T1=[ Number\*Number->Number] \* [Number\*Number->Number] -> T0 |
|  | Tf=[ Number\*Number->Number] \* Number -> T0 |
| Tx=Number | Tnum1=Number |
| T2=Number | T+=Number\*Number->Number |
|  | T\*=Number\*Number->Number |
|  | Tx= Number\*Number->Number |
|  | T2=T0 |

Tx=Number and Tx=Number\*Number - > Number

We get conflicting equation , so the expression is not well typed.

Part 3:

**Question 3.2.2**

Typing rule Define-type:

For every user-defined-type id  
 with component records record\_1 ... record\_n  
 with fields (field\_ij) (i in [1...n], j in [1..R\_i]), with types (type\_ij) (i in [1...n], j in [1..R\_i]):

If id : S

Then ( define-type id (record\_1 (field\_11 : type\_11 ... field\_1r1 : type\_1r1))... ) : S

Typing rule Type-case:  
 For every user-defined-type id  
 with component records record\_1 ... record\_n  
 with fields (field\_ij) (i in [1...n], j in [1..R\_i]),  
 val CExp,  
 body\_i for i in [1..n] sequences of CExp type expression S, P:  
 If id |- val : S id |- record\_i : S id |- body\_i : P

Then ( type-case id val (record\_1 (field\_11 ... field\_1r1) body\_1)... ) : P