Question 1

a.1. (foo (goo 0))

(define foo

(lambda (x) (display x) (newline) (+ x 2)))

(define goo

(lambda (x) (display 5) (newline) (+ x 1)))

normal-eval[(foo (goo 0 ))]

normal-eval[foo] ==> <closure (lambda (x) (display x) (newline) (+ x 2))>

sub[x, (goo 0), ((display x) (new line) (+ x 2)) ==> ((display (goo 0)) (newline) (+ (goo 0) 2))]

normal-eval[(display (goo 0)) (newline) (+ (goo 0) 2)]

normal-eval[(display (goo 0))]

normal-eval[display] ==> <primitive-procedure display>

normal-eval[(goo 0)] ==> <closure (lambda (x) (display 5) (newline) (+ x 1))>

sub[x, 0, ((display 5) (newline) (+ x 1)) ==> ((display 5) (newline) (+ 0 1))]

normal-eval[(display 5) (newline) (+ 0 1)]

normal-eval[(display 5)]

normal-eval[display] ==> <primitive-procedure display>

normal-eval[5] ==> 5

==> 5

normal-eval[newline] ==> <primitive-procedure newline>

normal-eval[(+ 0 1)]

normal-eval[+] ==> <primitive-procedure +>

normal-eval[0] ==> 0

normal-eval[1] ==> 1

==> 1

normal-eval[newline] ==> <primitive-procedure newline>

normal-eval[(+ (goo 0) 2)]

normal-eval[+] ==> <primitive-procedure +>

normal-eval[(goo 0)] ==> <closure (lambda (x) (display 5) (newline) (+ x 1))>

sub[x, 0, ((display 5) (newline) (+ x 1)) ==> ((display 5) (newline) (+ 0 1))]

normal-eval[(display 5) (newline) (+ 0 1)]

normal-eval[(display 5)]

normal-eval[display] ==> <primitive-procedure display>

normal-eval[5] ==> 5

==> 5

normal-eval[newline] ==> <primitive-procedure newline>

normal-eval[(+ 0 1)]

normal-eval[+] ==> <primitive-procedure +>

normal-eval[0] ==> 0

normal-eval[1] ==> 1

==> 1

normal-eval[2] ==> 2

==> 3

a.2.

under normal-eval it will be displayed:

5

1

5

3

(where 3 is the returned value in the end and all the rest are displayed by using the procedure display)

and under applicative-eval it will be displayed:

5

1

3

b. for example:

>(define f (lambda (x) (f x)))

>(define g (lambda (x) 5)))

And the expression (g (f 0)) using normal-eval will return the value 5, while in applicative-eval will enter to an infinite loop.

That's because in applicative-eval, every operand is being evaluated before substitution, in our case

(f 0) is evaluated over and over again. Whereas in normal-eval (f 0) is not being evaluated once­.

Question 2

a.1) (lambda (f n) (lambda (x) (\* n (f x))))

The leaves are \*,n,f,x:

1. {} |- \*:Number \* Number -> Number

2. { n:T1 } |- n:T1

3. { f:T2 } |- f:T2

4. { x:T3 } |- x:T3

Typing (f x)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {x:T3,f:T2} |
| \_f | f |
| \_e1 | x |
| \_s1 | T3 |
| \_s | T4 |

where \_s=T2=T3->T4

5. {x:T3,f:[T3->T4]} |- (f x):T4

Typing (\* n (f x))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {x:T3,f:[T3->T4],n:T1} |
| \_f | \* |
| \_e1 | n |
| \_e2 | (f x) |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

6. {x:T3,n:Number,f:[T3->Number]} |- (\* n (f x)):Number

Typing (lambda(x) (\* n (f x)))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {x:T3,f:[T3->Number],n:Number} |
| \_x1 | x |
| \_b1 | (\* n (f x)) |
| \_s1 | T3 |
| \_u1 | Number |

7.{n:Number,f:[T3->Number]} |- (lambda(x) (\* n (f x))):[T3]->Number

Typing (lambda (f n)(lambda(x) (\* n (f x))))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {n:T1,f:[T3->T4]} |
| \_x1 | f |
| \_x2 | n |
| \_b1 | (lambda(x) (\* n (f x))) |
| \_s1 | T3->Number |
| \_s2 | Number |
| \_u1 | T3->Number |

8. {} -| (lambda (f n)(lambda(x) (\* n (f x)))):[[T3->Number]\*Number]->[T3->Number]

well-typed מכיוון שהסביבה ריקה וכל משתנה מוגדר הביטוי אכן

a.2) ((lambda(f n) (lambda (x) (\* n (f x)))) f 3 )

The leaves are: \*, n , f, x

1. {} |- (lambda (f n) (lambda(x) (\* n ( f x)))):

[[T3 -> Number]\*Number] -> [T3 -> Number]

1. Reanming: ((lambda(f n) (lambda (x) (\* n (f x)))) f1 3 )
2. {f1:T5} |- f1:T5
3. {} |- 3:Number

Typing ((lambda(f n) (lambda (x) (\* n (f x)))) f 3 )

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {f1:T5} |
| \_f | ((lambda(f n) (lambda (x) (\* n (f x)))) |
| \_e1 | f |
| \_e2 | 3 |
| \_s1 | [T3->Number] |
| \_s2 | Number |
| \_s | T3->Number |

s1=T5=[T3->Number]

5. {f:[T3->Number]} |- ((lambda(f n) (lambda (x) (\* n (f x)))) f 3 )

אך מכיוון שהסביבה אינה ריקה - הביטוי אינו Well-Typed.

a. 3) (define dg (lambda (number)(if (< number 10)1 (+ (dg (quotient number 10)) 1))))

The leaves are: 1,10,<, number,+,dg,quotient,if

1. {} |- 1: Number
2. {} |- 10:Number
3. {} |- <:Number\*Number -> Boolean
4. {number:T0} |- number:T0
5. {} |- +:Number\*Number -> Number
6. {dg:T1} |- dg:T1
7. {} |- quotient:Number\*Number -> Number
8. {if:T2} |- if:T2

Typing (quotient number 10)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {number:T0} |
| \_f | quotient |
| \_e1 | number |
| \_e2 | 10 |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

1. {number:Number} |- (quotient number 10): Number

Typing (dg (quotient number 10))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {number:Number, dg:T1} |
| \_f | dg |
| \_e1 | (quotient numebr 10) |
| \_s1 | Number |
| \_s | T3 |

s=T1=Number -> T3

1. {number:Number, dg:Number ->T3} |- (dg (quotient number 10)): T3

Typing (+ (dg (quotient number 10)) 1)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {number:Number, dg:Number -> T3} |
| \_f | + |
| \_e1 | (dg (quotient numebr 10)) |
| \_e2 | 1 |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

T3 =Number.

1. {number:Number,dg:Number->Number} |- (+ (dg (quotient number 10)) 1): Number

Typing (< number 10)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {number:Number} |
| \_f | < |
| \_e1 | Number |
| \_e2 | 10 |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

1. {number:Number} |- (< number 10):Boolean

Typing (if (< number 10) 1 (+ (dg (quotient number 10)) 1))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {number:Number, dg:Number -> Number} |
| \_p | (< number 10) |
| \_c | 1 |
| \_a | (+ (dg (quotient number 10)) 1) |
| \_s | T4 |

\_s=T4=NumberNumber = Number

1. {number:Number,dg:Number->Number} |- (if (< number 10) 1 (+ (dg (quotient number 10)) 1)):Number

Typing (lambda(number) (if (< number 10) 1 (+ (dg (quotient number 10)) 1)))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {dg:Number -> Number} |
| \_x1 | number |
| \_b1 | (if (< number 10) 1 (+ (dg (quotient number 10)) 1)) |
| \_s1 | Number |
| \_u1 | Number |

1. {dg:Number->Number} |-(lambda(number) (if (< number 10) 1 (+ (dg (quotient number 10)) 1))): [Number -> Number]
2. {} |- (lambda(number) (if (< number 10) 1 (+ (dg (quotient number 10)) 1))): Number

.well-typed מכיוון שהסביבה ריקה וכל משתנה מוגדר הביטוי אכן

a.4) (+ (lambda(x) 5) x)

The leaves are +,5,x:

1. 1. {} |- +:Number \* Number -> Number
2. { x:T1 } |- x:T1
3. {} |- Number |- 5:Number
4. Reanaming: (+ (labmda(x1) 5) x)
5. {x1:T2} |- x1:T2

Typing (lambda(x) 5)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {x1:T2} |
| \_f | x1 |
| \_e1 | 5 |
| \_s1 | T2 |
| \_s | Number |

6. {x1:T2} |- (lambda(x1) 5): [T2 -> Number]

Typing (+ (lambda(x) 5) x)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {x1:T2, x:T1} |
| \_f | + |
| \_e1 | (lambda(x1) 5) |
| \_e2 | x |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

7. {x1:T2, x:T1} :(+ (lambda(x) 5) x) . [T2->Number] \* Number -> Number.

אך מכיוון שהסביבה אינה ריקה - הביטוי אינו Well-Typed, והפרוצדורה + מקבלת בארגומנט השמאלי פרוצדורה

(T2->Number) במקום לקבל רק (Number).

b. Typing rule Application of procedures with one parameter:

For every type environment \_Tenv,

expressions \_f, \_e and

type expressions \_S1, ... , \_Sm, \_S:

If \_Tenv |- \_f:[ \_S1 union \_S2 union ... union \_Sm -> \_S]

\_Tenv |- \_e:Si where

Then \_Tenv |- (\_f \_e):\_S

c. (define number\_proc

(lambda(par) (if (number? par) (+ par 1) (+ (par 1) 1))))

(number\_proc 10)

The leaves are: number\_proc, par, number? , +, 1, if

1. {} |- 1:Number

2. {} |- Number?:[Number->Boolean]

3. {} |- +:[Number\*Number->Number]

4. {number\_proc:T0} |- number\_proc:T0

5. {par:T1} |- par:T1

6. {if:T2} |- if:T2

Typing (par 1)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {par:T1} |
| \_f | par |
| \_e1 | 1 |
| \_s1 | Number |
| \_s | T3 |

\_s=T1=[Number->T3]

7. {par:T1} |- (par 1):T3

Typing (+ par 1)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {par:[Number->T3]} |
| \_f | + |
| \_e1 | par |
| \_e2 | 1 |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

8.{par:Number U [Number->number] |- (+ par 1):Number

Typing (number? par)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {par:[Number ->Number]Number Number } |
| \_f | number? |
| \_e1 | par |
| \_s1 | [Number ->Number]Number |
| \_s | Boolean |

9. {par:[Number ->Number]Number Number } |- (number? par):Boolean.

Typing (+ (par 1) 1)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {par:[Number ->Number]Number Number } |
| \_f | + |
| \_e1 | (par 1) |
| \_e2 | 1 |
| \_s1 | Number |
| \_s2 | Number |
| \_s | Number |

10. {par:[Number ->Number]Number Number } |- (+ (par 1) 1) :Number.

Typing (if (number? par) (+ par 1) (+ (par 1) 1))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {par:[Number ->Number]Number Number } |
| \_p | (number? par) |
| \_c | (+ par 1) |
| \_a | (+ (par 1) 1) |
| \_s | T4 |

\_s=T4=[Number->Number]Number = Number.

11. {par:[Number ->Number]Number Number } |-

(if (number? par) (+ par 1) (+ (par 1) 1)):Number.

Typing (lambda(par) (if (number? par) (+ par 1) (+ (par 1) 1)))

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {} |
| \_x1 | par |
| \_b1 | (if (number? par) (+ par 1) (+ (par 1) 1)) |
| \_s1 | Number U [Number->Number] |
| \_u1 | Number |

12. {} |- (lambda(par) (if (number? par) (+ par 1) (+ (par 1) 1))) :

[Number U [Number->Number] -> Number]

number-proc=T0=Number [Number -> Number] .

הביטוי (define number\_proc) אכן well-typed

Typing (number\_proc 10)

|  |  |
| --- | --- |
| Var | Replacement |
| \_Tenv | {} |
| \_f | number\_proc |
| \_e1 | 10 |
| \_s1 | Number |
| \_s2 | [Number->Number] |
| \_s | Number |

13. {} |- (number\_proc 10):Number

גם ביטוי זה הינו well-typed

Question 3

1. (lambda (f) (f ((lambda (g y) (+ 1 (g y))) + 10)))

Step 1:

|  |  |
| --- | --- |
| Expression | Var |
| (lambda (f) (f ((lambda (g y) (+ 1 (g y))) + 10))) | T0 |
| (f ((lambda (g y) (+ 1 (g y))) + 10)) | T1 |
| ((lambda (g y) (+ 1 (g y))) + 10) | T2 |
| (lambda (g y) (+ 1 (g y))) | T3 |
| (+ 1 (g y)) | T4 |
| (g y) | T5 |
| g | T6 |
| y | T7 |
| + | T8 |
| 10 | T9 |
| 1 | T10 |
| f | T11 |

Step 2:

|  |  |
| --- | --- |
| Expression | Equation |
| 1 | T10:=Number |
| 10 | T9:=Number |
| + | T8:=[Number\*Number->Number] |
| (g y) | T6:=[T7->T5] |
| (+ 1 (g y)) | T8:=[T10\*T5->T4] |
| (lambda (g y) (+ 1 (g y))) | T3:=[T6\*T7->T4] |
| ((lambda (g y) (+ 1 (g y))) + 10) | T3:=[T8\*T9->T2] |
| (f ((lambda (g y) (+ 1 (g y))) + 10)) | T11:=[T2->T1] |
| (lambda(f) (f ((lambda (g y) (+ 1 (g y))) + 10))) | T0:=[T11->T1] |

Step 3:

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [T11 -> T1] | T0 := [T11 -> T1] |
|  | T11 := [T2 ->T1] |
|  | T3 := [[T8 \* T9] -> T2] |
|  | T3 := [[T6 \* T7] ->T4] |
|  | T8 := [[T10 \* T5] -> T4] |
|  | T6 :=[T7 -> T5] |
|  | T8 := [Number \* Number -> Number] |
|  | T9:=Number |
|  | T10:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [T11 -> T1] | T11 := [T2 ->T1] |
| T11 := [T2 ->T1] | T3 := [[T8 \* T9] -> T2] |
|  | T3 := [[T6 \* T7] ->T4] |
|  | T8 := [[T10 \* T5] -> T4] |
|  | T6 :=[T7 -> T5] |
|  | T8 := [Number \* Number -> Number] |
|  | T9:=Number |
|  | T10:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T3 := [[T8 \* T9] -> T2] |
| T11 := [T2 ->T1] | T3 := [[T6 \* T7] ->T4] |
| T3 := [[T8 \* T9] -> T2] | T8 := [[T10 \* T5] -> T4] |
|  | T6 :=[T7 -> T5] |
|  | T8 := [Number \* Number -> Number] |
|  | T9:=Number |
|  | T10:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T3 := [[T6 \* T7] ->T4] |
| T11 := [T2 ->T1] | T8 := [[T10 \* T5] -> T4] |
| T3 := [[T8 \* T9] -> T2] | T6 :=[T7 -> T5] |
|  | T8 := [Number \* Number -> Number] |
|  | T9:=Number |
|  | T10:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T8 := [[T10 \* T5] -> T4] |
| T11 := [T2 ->T1] | T6 :=[T7 -> T5] |
| T3 := [[T8 \* T9] -> T2] | T8 := [Number \* Number -> Number] |
| T8 := [[T10 \* T5] -> T4] | T9:=Number |
|  | T10:=Number |
|  | T6:=T8 |
|  | T7:=T9 |
|  | T4:=T2 |



|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T6 :=[T7 -> T5] |
| T11 := [T2 ->T1] | T8 := [Number \* Number -> Number] |
| T3 := [[[[T10 \* T5] -> T4] \* T9] -> T2] | T9:=Number |
| T8 := [[T10 \* T5] -> T4] | T10:=Number |
| T6 :=[T7 -> T5] | T6:=T8 |
|  | T7:=T9 |
|  | T4:=T2 |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T8 := [Number \* Number -> Number] |
| T11 := [T2 ->T1] | T9:=Number |
| T3 := [[[[T10 \* T5] -> T4] \* T9] -> T2] | T10:=Number |
| T8 := [[T10 \* T5] -> T4] | T6:=T8 |
| T6 :=[T7 -> T5] | T7:=T9 |
|  | T4:=T2 |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T9:=Number |
| T11 := [T2 ->T1] | T10:=Number |
| T3 := [[[[T10 \* T5] -> T4] \* T9] -> T2] | T6:=T8 |
| T8 := [[T10 \* T5] -> T4] | T7:=T9 |
| T6 :=[T7 -> T5] | T4:=T2 |
|  | T10:=Number |
|  | T5:=Number |
|  | T4:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T9:=Number |
| T11 := [T2 ->T1] | T10:=Number |
| T3 := [[[[T10 \* T5] -> T4] \* Number] -> T2] | T6:=T8 |
| T8 := [[T10 \* T5] -> T4] | T7:=T9 |
| T6 :=[T7 -> T5] | T4:=T2 |
| T9:=Number | T10:=Number |
|  | T5:=Number |
|  | T4:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0 := [[T2->T1] -> T1] | T10:=Number |
| T11 := [T2 ->T1] | T6:=T8 |
| T3 := [[[[Number \* T5] -> T4] \* Number] -> T2] | T7:=T9 |
| T8 := [[Number \* T5] -> T4] | T4:=T2 |
| T6 :=[T7 -> T5] | T10:=Number |
| T9:=Number | T5:=Number |
| T10:=Number | T4:=Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[T2->T1] -> T1] | T6:=T8 (T6 := Number -> Number) |
| T11:= [T2 ->T1] | T7:=T9 |
| T3:= [[[[Number \* T5] -> T4] \* Number] -> T2] | T4:=T2 |
| T8:= [[Number \* T5] -> T4] | T10:=Number |
| T6:=[T7 -> T5] | T5:=Number |
| T9:=Number | T4:=Number |
| T10:=Number |  |
|  |  |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[T2->T1] -> T1] | T7:=T9 |
| T11:= [T2 ->T1] | T4:=T2 |
| T3:= [[[[Number \* T5] -> T4] \* Number] -> T2] | T10:=Number |
| T8:= [[Number \* T5] -> T4] | T5:=Number |
| T6:=[T7 -> T5] | T4:=Number |
| T9:=Number | T7 := Number |
| T10:=Number | T5 := Number |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[T2->T1] -> T1] | T7:=T9 |
| T11:= [T2 ->T1] | T4:=T2 |
| T3:= [[[[Number \* T5] -> T4] \* Number] -> T2] | T10:=Number |
| T8:= [[Number \* T5] -> T4] | T5:=Number |
| T6:=[Number -> T5] | T4:=Number |
| T9:=Number | T7:= Number |
| T10:=Number | T5:= Number |
| T7:=T9 |  |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[T2->T1] -> T1] | T4:=T2 |
| T11:= [T2 ->T1] | T10:=Number |
| T3:= [[[[Number \* T5] -> T2] \* Number] -> T2] | T5:=Number |
| T8:= [[Number \* T5] -> T2] | T4:=Number |
| T6:=[Number -> T5] | T7:= Number |
| T9:=Number | T5:= Number |
| T10:=Number |  |
| T7:=T9 |  |
| T4:=T2 |  |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[T2->T1] -> T1] | T5:=Number |
| T11:= [T2 ->T1] | T4:=Number |
| T3:= [[[[Number \* Number] -> T2] \* Number] -> T2] | T7:= Number |
| T8:= [[Number \* Number] -> T2] | T5:= Number |
| T6:=[Number -> Number] |  |
| T9:=Number |  |
| T10:=Number |  |
| T7:=T9 |  |
| T4:=T2 |  |
| T5:=Number |  |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[T2->T1] -> T1] | T7:= Number |
| T11:= [T2 ->T1] | T5:= Number |
| T3:= [[[[Number \* Number] -> T2] \* Number] -> T2] |  |
| T8:= [[Number \* Number] -> T2] |  |
| T6:=[Number -> Number] |  |
| T9:=Number |  |
| T10:=Number |  |
| T7:=T9 |  |
| T2:=Number |  |
| T5:=Number |  |
| T4:=Number |  |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[Number->T1] -> T1] | T7:= Number |
| T11:= [Number ->T1] | T5:= Number |
| T3:= [[[[Number \* Number] -> Number] \* Number] -> Number] |  |
| T8:= [[Number \* Number] -> Number] |  |
| T6:=[Number -> Number] |  |
| T9:=Number |  |
| T10:=Number |  |
| T9:=Number |  |
| T2:=Number |  |
| T5:=Number |  |
| T4:=Number |  |
| T7:= Number |  |

|  |  |
| --- | --- |
| Substitution | Equation |
| T0:= [[Number->T1] -> T1] |  |
| T11:= [Number ->T1] |  |
| T3:= [[[[Number \* Number] -> Number] \* Number] -> Number] |  |
| T8:= [[Number \* Number] -> Number] |  |
| T6:=[Number -> Number] |  |
| T9:=Number |  |
| T10:=Number |  |
| T9:=Number |  |
| T2:=Number |  |
| T5:=Number |  |
| T4:=Number |  |
| T7:= Number |  |

The type inference succeeds. The inferred type is: [[Number -> T1] -> T1]

2. (lambda(f)

(lambda(y)

(y (f f))))

Variables:

|  |  |
| --- | --- |
| (lambda(f)(lambda(y)(y (f f)))) | T0 |
| (lambda(y)(y (f f))) | T1 |
| (y (f f)) | T2 |
| (f f) | T3 |
| f | T4 |

Equations definition:

|  |  |
| --- | --- |
| Expression | Equation |
| (f f) | T4:=T4 -> T3 |
| ( y (f f)) | T5 := T3 -> T2 |
| (lambda (y) (y (f f))) | T1 := T5 -> T2 |
| (lambda (f) (lambda (y) (y (f f)))) | T0 := T4 -> T1 |

Step 1:

|  |  |
| --- | --- |
| Equation | Substitution |
| T4:=T4 -> T3 | T4:=T4 -> T3 |
| T5 := T3 -> T2 |  |
| T1 := T5 -> T2 |  |
| T0 := T4 -> T1 |  |

Step 2:

|  |  |
| --- | --- |
| Equation | Substitution |
| T5 := T3 -> T2 | T4:=T4 -> T3 |
| T1 := T5 -> T2 | T5 := T3 -> T2 |
| T0 := T4 -> T1 |  |

Step 3:

|  |  |
| --- | --- |
| Equation | Substitution |
| T1 := T5 -> T2 | T4:=T4 -> T3 |
| T0 := T4 -> T1 | T5 := T3 -> T2 |

Step 4:

|  |  |
| --- | --- |
| Equation | Substitution |
| T0 := T4 -> T1 | T4:=T4 -> T3 |
|  | T5 := T3 -> T2 |
|  | T1 := T5 -> T2 |

Step 5:

|  |  |
| --- | --- |
| Equation | Substitution |
|  | T4:=T4 -> T3 |
|  | T5 := T3 -> T2 |
|  | T1 := (T3 -> T2) -> T2 |
|  | T0 := T4 -> T1 |

Step 6:

|  |  |
| --- | --- |
| Equation | Substitution |
|  | T4:=T4 -> T3 |
|  | T5 := T3 -> T2 |
|  | T1 := (T3 -> T2) -> T2 |
|  | T0 := (T4 -> T3)-> (T3 -> T2) -> T2 |

A circular substitution occurred? Output FAIL.

3. (lambda (x) (+ x (x 1)))

Step 1:

|  |  |
| --- | --- |
| Expression | Var |
| (lambda (x) (+ x (x 1))) | T0 |
| (+ x (x 1)) | T1 |
| (x 1) | T2 |
| + | T3 |
| x | T4 |
| 1 | T5 |

Step 2:

|  |  |
| --- | --- |
| Expression | Equation |
| 1 | T5:=Number |
| + | T3:=[Number\*Number->Number] |
| (x 1) | T4:=[Number -> T2] |
| (+ x (x 1)) | T3:=[T4\*T2->T1] |
| (lambda(x) (+ x (x 1))) | T0:=[T4->T1] |

Step 3:

|  |  |
| --- | --- |
| Equation | Substitution |
| T0:=[T4->T1] | T0:=[T4->T1] |
| T3:=[T4\*T2->T1] |  |
| T3:=[Number\*Number->Number] |  |
| T4:[Number->T2] |  |
| T5:=Number |  |

|  |  |
| --- | --- |
| Equation | Substitution |
| T3:=[T4\*T2->T1] | T0:=[T4->T1] |
| T3:=[Number\*Number->Number] | T3:=[T4\*T2->T1] |
| T4:[Number->T2] |  |
| T5:=Number |  |

|  |  |
| --- | --- |
| Equation | Substitution |
| T3:=[Number\*Number->Number] | T0:=[T4->T1] |
| T4:[Number->T2] | T3:=[T4\*T2->T1] |
| T5:=Number |  |

|  |  |
| --- | --- |
| Equation | Substitution |
| T3:=[Number\*Number->Number] | T0:=[T4->T1] |
| T4:[Number->T2] | T3:=[T4\*T2->T1] |
| T5:=Number |  |

|  |  |
| --- | --- |
| Equation | Substitution |
| T4:[Number->T2] | T0:=[T4->T1] |
| T5:=Number | T3:=[T4\*T2->T1] |
| T4:=Number |  |
| T2:=Number |  |
| T1:=Number |  |

|  |  |
| --- | --- |
| Equation | Substitution |
| T4:[Number->T2] | T0:=[[Number->T2]->T1] |
| T5:=Number | T3:=[[Number->T2]\*T2->T1] |
| T4:=Number | T4:[Number->T2] |
| T2:=Number |  |
| T1:=Number |  |

|  |  |
| --- | --- |
| Equation | Substitution |
| T5:=Number | T0:=[[Number->T2]->T1] |
| T4:=Number | T3:=[[Number->T2]\*T2->T1] |
| T2:=Number | T4:[Number->T2] |
| T1:=Number | T5:=Number |

|  |  |
| --- | --- |
| Equation | Substitution |
| T4:=Number | T0:=[[Number->T2]->T1] |
| T2:=Number | T3:=[[Number->T2]\*T2->T1] |
| T1:=Number | T4:[Number->T2] |
|  | T5:=Number |

We receive that one side is atomic and the other is composite but not with the same type constructor therefore we the output should FAIL.

Question 4

a.

Concrete Syntax:

<scheme-exp> <exp> | <define>

<exp> <atomic> | <composite>

<composite> <special> | <form>

<special> <lambda> | <quote> | <cond> | <if> | <case>

<case> ‘(‘ ‘case’ <exp> <exp> ‘(‘ <case-clause>\* ‘)’ <case-else-clause>

<case-clause>’(‘ <datum>\* ‘)’ <exp>+

<datum> <exp>

<case-else-clause> ’(‘ ’else’ <exp>+ ‘)’

Abstract Sytanx:

<scheme-exp>:

Kinds: <exp>, <define>

<exp>:

Kinds: <atomic>, <composite>

<composite>:

Kinds: <special>, <form>

<special>:

Kinds: <lambda>, <quote>, <cond>, <if> , <case>

<case> :

Components:

Key: <exp>

Predicate: <exp>

Case-clauses: <case-clauses> ; Amount >= 1 ; Ordered.

Else-clause: <case-else-clause>

<case-clause>:

components:

Datum: <exp> ; Amount >= 1.

Consequence: <exp> ; Amount >= 1 ; Ordered.

<case-else-clause>:

components:

Consequence: <exp>; Amount >=1 ; Ordered.

Question 5

a)

1. (let ((gcd (lambda(a b)

2. (if (= b 0)

3. a

4. (gcd b (modulo a b) )))))

5. (gcd (+ (+ 3 3) (+ 3 3)) 8))

|  |  |  |  |
| --- | --- | --- | --- |
| Binding instance | Binding instance, line# | Scope | Bound occurrences,line # |
| gcd | 1 | 1-5 | 4,5 |
| a | 1 | 1-4 | 3,4 |
| b | 1 | 1-4 | 2,4 |

b)

אחת הבעיות במודל ההחלפה (כפי שנידון בכיתה) הוא שלא ניתן לחשב פונקציות רקורסיביות מקומיות שהוגדרו זה עתה ע"י ה let. מכיוון שבסביבה הגלובלית לא הוגדר המשתנה המייצג את הפונקציה, בעת קריאה רקורסיבית לפונקציה נקבל שגיאה.

ולכן נשתמש באופרטור המיוחד לכך: letrec ובכך יהיה ידוע המשתנה המצייג את הפונקציה בscope אשר הוגדר ע"י ה-letrec.

c)

Applicative Eval.

applicative-eval[(let ((gcd (lambda(a b gcdnew) (if (= b 0) a (gcdnew b (modulo a b)

gcdnew))))) (gcd (+ (+ 3 3) (+ 3 3)) 8 gcd))]

applicative-eval[(define gcd

(lambda (a b gcdnew)

(if (= b 0)

a

(gcdnew b (modulo a b) gcdnew)))))]

applicative-eval[(lambda(a b gcdnew) (if (= b 0) a (gcdnew b (modulo a b) gcdnew)))))]

==> <Closure(lambda (a b gcdnew) (if (= b 0) a (gcdnew b (modulo a b) gcdnew))))>

applicative-eval[(gcd (+ ( + 3 3) (+ 3 3)) 8 gcd))]

applicative-eval[gcd]==><Closure (lambda a b gcdnew) (if (= b 0) a (gcdnew b

(modulo a b) gcdnew)))))>

applicative-eval[(+ ( + 3 3) (+ 3 3))]

applicative-eval[+]==> #<primitive-procedure +>

applicative-eval[(+ 3 3)]

applicative-eval[+] ==> #<primitive-procedure +>

applicative-eval[3] ==> 3

applicative-eval[3] ==> 3

==>6

applicative-eval[(+ 3 3)]

applicative-eval[+] ==> #<primitive-procedure+>

applicative-eval[3] ==> 3

applicative-eval[3] ==> 3

==>6

==>12

applicative-eval[8] ==> 8

applicative-eval[gcd] ==> <Closure (lambda a b gcdnew) (if (= b 0) a (gcdnew b

(modulo a b) gcdnew)))))>

Substitution!

sub(a, 12 ⇒ (if (= b 0) 12 (gcdnew b (modulo 12 b) gcdnew)))))

sub(b , 8 ⇒ (if (= 8 0) 12 (gcdnew 8 (modulo 12 8) gcdnew)))))

sub(gcdnew, gcd ⇒ (if (= 8 0) 12 (gcd 8 (modulo 12 8) gcd)))))

reduce:

applicative-eval[(if (= 8 0) 12 (gcd 8 (modulo 12 8) gcd)))))]

applicative-eval[if] ⇒ #<special-form>

applicative-eval[(= 8 0)]

applicative-eval[=] ==> <primitive-procedure =>

applicative-eval[8] ==> 8

applicative-eval[0] ==> 0

=> #f

applicative-eval[12] ==> 12

application-eval[(gcd 8 (modulo 12 8) gcd))]

application-eval[gcd] ==> <Closure (lambda a b gcdnew) (if (= b 0) a

(gcdnew b (modulo a b) gcdnew)))))>

application-eval[8] ==> 8

application-eval[modulo 12 8]

application-eval[modulo] ==> <procedure %>

application-eval[12] ==> 12

application-eval[8] ==> 8

==> 4

application-eval[gcd] ==> <Closure (lambda a b gcdnew) (if (= b 0) a

(gcdnew b (modulo a b) gcdnew)))))>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sub(a, 8 ⇒ (if (= b 0) 8 (gcdnew b (modulo 8 b) gcdnew)))))

sub(b , 4 ⇒ (if (= 4 0) 8 (gcdnew 4 (modulo 8 4) gcdnew)))))

sub(gcdnew, gcd ⇒ (if (= 4 0) 8 (gcd 4 (modulo 8 4) gcd)))))

reduce:

applicative-eval[(if (= 4 0) 8 (gcd 4 (modulo 8 4) gcd)))))]

applicative-eval[if] ⇒ #<special-form>

applicative-eval[(= 4 0)]

applicative-eval[=] ==> <primitive-procedure =>

applicative-eval[4] ==> 4

applicative-eval[0] ==> 0

=> #f

applicative-eval[8] ==> 8

application-eval[(gcd 4 (modulo 8 4) gcd))]

application-eval[gcd] ==> <Closure (lambda a b gcdnew) (if (= b 0) a

(gcdnew b (modulo a b) gcdnew)))))>

application-eval[4] ==> 4

application-eval[modulo 8 4]

application-eval[modulo] ==> <procedure %>

application-eval[8] ==> 8

application-eval[4] ==> 4

==> 0

application-eval[gcd] ==> <Closure (lambda a b gcdnew) (if (= b 0) a

(gcdnew b (modulo a b) gcdnew)))))>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sub(a, 4 ⇒ (if (= b 0) 4 (gcdnew b (modulo 4 b) gcdnew)))))

sub(b , 0 ⇒ (if (= 0 0) 4 (gcdnew 0 (modulo 4 0) gcdnew)))))

sub(gcdnew, gcd ⇒ (if (= 0 0) 4 (gcd 0 (modulo 4 0) gcd)))))

reduce:

applicative-eval[(if (= 0 0) 4 (gcd 0 (modulo 4 0) gcd)))))]

applicative-eval[if] ⇒ #<special-form>

applicative-eval[(= 0 0)]

applicative-eval[=] ==> <primitive-procedure =>

applicative-eval[0] ==> 0

applicative-eval[0] ==> 0

=> #t

applicative-eval[4] ==> 4

⇒ 4. (final answer)

e.

(letrec ((choose (lambda (n k)

(cond ((= k 0) 1)

((= k n) 1)

(else (+ (choose (- n 1) k) (choose (- n 1) (- k 1)))))))) (choose 6 4))