Assignment 4

b. we want to prove that (pipe$ f1..fn c) is equivalent to (c (pipe f1..fn))

base case- n=1

(pipe$ f1$ c1) = (c1(lambda(x c2) (f1$ x c2)))=c1(f1$ x c2)=(c1(f1$))

(c1(pipe (f$))) = (c1 (f$))

While f$ is the cps of f

induction hypothesis- n = k

(pipe$ f1..fk c) = (c (pipe f1...fk))

inductive step - n = k+1

need to prove - (pipe$ f1...f(k+1) c) = (c (pipe f1...f(k+1)))

lst-fun= f1...f(k+1)

- ⇨ (pipe$ lst-fun c1) =

  (pipe$ (cdr lst-fun) (lambda (res) (compose$ (car lst-fun) res c1)))))) =

  ((lambda (res) (compose$ (car lst-fun) res c1))(pipe$ (cdr lst-fun c1)))))=

  ((lambda (res) (compose$ (car lst-fun) res c1))c1(pipe(cdr lst-fun))**=//induction**

  C1(compose (car lst-fun) (pipe(cdr lst-fun)))**=// induction**

- ⇨ (c1 (pipe lst-fun))=

  (c1 (compose (car lst-fun) pipe (cdr lst-fun))

we will prove that (compose $ f$ g$ c) is equivalent to (c (compose f g)):

c(Compose f g)=c(g(f x))

we assume that f$ x c=c(f x) , g$ y c=c(g y).

(compose$ f$ g$ c1)=c1(lambda (x c2)(f$ x lambda (res)(g$ res c2))=

c1(lambda (res)(g$ res c2)(f x)= c1(c2(g(f(x))=c(g(f x) as needed.

Question 2-

d. we will use each of the following procedures for a different purpose-

reduce1-lzl – when we need to reduce all the elements of the lazy list into a single value when the lazy list are   -

Examples : calculating the sum or product of all elements in a lazy list, finding the maximum or minimum element in a lazy list.

Reduce2-lzl- when you only need to reduce the first n elements of the lazy list or when dealing with large or infinite lazy lists where processing the entire list isn't feasible. Examples - Calculating the sum of the first n elements in a lazy list , finding the average of the first n elements in a lazy list.

Reduce3-lzl- when you need to process and potentially consume the lazy list incrementally while producing a lazy list of results.

Examples- Generating a lazy list where each element is the cumulative sum up to that point.producing a lazy list of running averages as elements are processed.

G.

Advantages for 'generate-pi-approximations' implementation w.r.t. the pi-sum implementation taught in class are:

1. **Lazy evaluation :** generate-pi-approximations uses lazy lists, which means it generates approximations on demand. This is memory efficient because it doesn't compute more terms than necessary.

2. **Intermediate Results :** It allows access to intermediate results of the approximation. You can get the first few terms without computing the entire sum.

3. Memory use: generate-pi-approximations uses lazy lists which means it computes the next element only when needed and don't waste valuable memory space.

Disadvantages:

1. **Complex**: The implementation is more complex, involving multiple higher-order functions and lazy list manipulations. This can make the code harder to understand and maintain.
2. **Direct calculation:** This implementation directly sums the terms without the additional overhead of lazy evaluation, which can make it faster for small inputs.

Question 3-

3.1

1. unify[x(y(y), T, y, z, k(K), y), x(y(T), T, y, z, k(K), L)]

a. [=x(y(y), T, y, z, k(K), y) x(y(T), T, y, z, k(K), L)]

sub: {}

b. picking one equation: [=x(y(y), T, y, z, k(K), y) x(y(T), T, y, z, k(K), L)]

c. **case 3-**

d. [y(y) = y(T), T=T, y=y, z=z, k(K)=k(K), y=L]

sub: {}

e. picking one equation: y(y) = y(T)

[T=T, y=y,z=z k(K)=k(K), y=L]

sub: {}

f. **case 3 –**

g. [y= T]

h. picking one equation: y= T

i. [T=T, y=y, z=z, k(K)=k(K), y=L]

sub:{}*{y= T}= {y= T}

j. picking one equation: T=T

[y=y,z=z, k(K)=k(K), y=L]

k. sub:{y= T}

l. picking one equation: y=y

[z=z, k(K)=k(K), y=L]

m. {y=T} * {y=y} = {y = T}

picking one equation: z=z

{y=T}

n. picking one equation: k(K)=k(K)

[y=L]

{y =T}

case 3 –

o. [K= K]

p. {y= T} * {K=K} = {y = T}

q. picking one equation: [y=L]

[]

Sub:{y = T}

r. sub: {y = T} * {y = L} = {y = T, L = T}

2. unify[f(a, M, f, F, Z, f, x(M)), f(a, x(Z), f, x(M), x(F), f, x(M))]

a. [=f(a, M, f, F, Z, f, x(M)), f(a, x(Z), f, x(M), x(F), f, x(M))]

sub: {}

b. picking one equation: [=f(a, M, f, F, Z, f, x(M)), f(a, x(Z), f, x(M), x(F), f, x(M))]

c. **case 3-**

d. [a=a, M= x(Z),f=f,F=x(M),Z = x(F),f=f, x(M)= x(M)]

sub: {}

e. picking one equation: a=a

[M= x(Z),f=f,F=x(M),Z = x(F),f=f, x(M)= x(M)]

sub: {}

f. **case 1-**

g. picking one equation: M= x(Z)

[f=f,F=x(M),Z = x(F),f=f, x(M)= x(M)]

sub: {}

h. case 3 – **FAIL NOT THE SAME STRUCTURE**

3. unify[t(A, B, C, n(A, B, C),x, y), t(a, b, c, m(A, B, C), X, Y)]

a) [=t(A, B, C, n(A, B, C),x, y), t(a, b, c, m(A, B, C), X, Y)]
   sub: {}
b) picking one equation: [=t(A, B, C, n(A, B, C),x, y), t(a, b, c, m(A, B, C), X, Y)]
c) case 3-
d) [A=a, B=b, C=c, n(A,B,C) =m(A,B,C), X=X, Y=Y]
e) picking one equation: A=a
f) case 2
g) [B=b, C=c, n(A,B,C) =m(A,B,C), X=X, Y=Y]
h) sub: {A=a}
i) picking one equation: B=b
j) case 2
k) [C=c, n(A,B,C) =m(A,B,C), X=X, Y=Y]
l) sub: {A=a} * {B=b} = {A=a, B=b}
m) picking one equation: C=c
n) case 2
o) [n(A,B,C) =m(A,B,C), X=X, Y=Y]
p) sub:{A=a, B=b}* {C=c} = {A=a, B=b, C=c}
q) picking one equation: n(A,B,C) =m(A,B,C)
   n could not be m because those are actual values and not generic variables.

4. unify[z(a(A, x, Y), D, g), z(a(d, x, g), g, Y)]

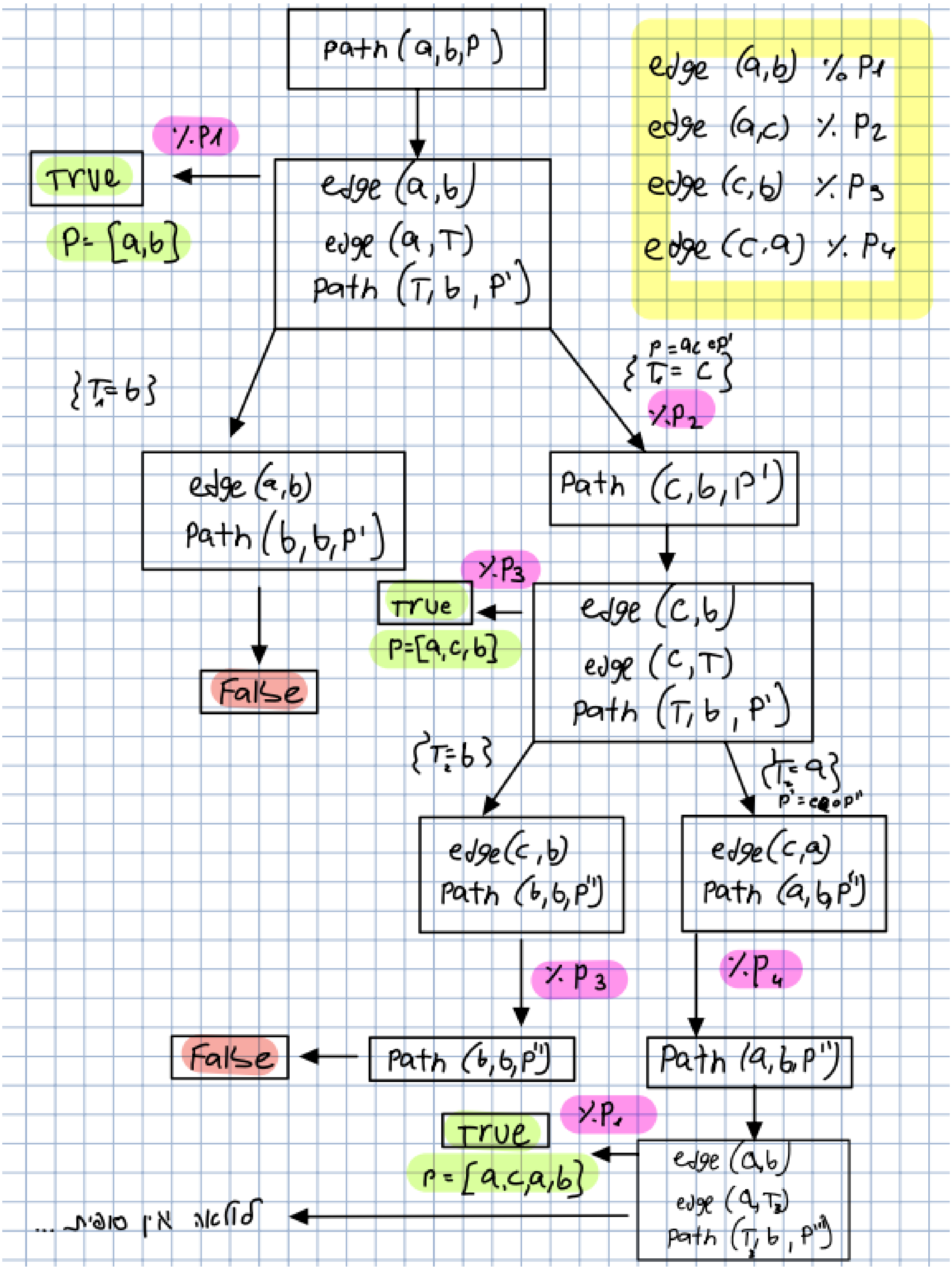
a) [= z(a(A,x,Y), D,g), z(a(d,x,g),g,Y)]
b) picking one equation: [= z(a(A,x,Y), D,g), z(a(d,x,g),g,Y)]
   sub: {}
c) case 3-
d) [a(A,x,Y)= a(d,x,g), D=g, g=Y]
   sub: {}
e) picking one equation: a(A,x,Y)= a(d,x,g)

```
f) case 3
g) [A=d,x=x,Y=g]
h) Sub: {A=d, Y=g}
i) picking one equation: D=g
j) case 2
k) [g=Y]
l) Sub: {A=d, Y=g}* {D=g} = (A=d, Y=g,D=g}
m) picking one equation: g=Y
n) case 2-
o) Sub: {A=d, Y=g}* {g=g} = (A=d, Y=g,D=g}
```

3.3

The proof tree of the query ?- path(a,b, P) is infinite because the tree contains the circle (a,c,a) and contains the edge (a,b) so the algorithm can perform the circle in infinite number of times before taking the edge (a,b).

In addition the tree is a success tree because there is at least One branch that leads to success.(example: (a,b) is a branch that leads to success).


edge (a,b) % P1
edge (a,c) % P2
edge (c,b) % P3
edge (c,a) % P4
path (a,b,P)
edge (a,b)
edge (a,T)
path (T,b,P')
% P1
True
P= [a,b]
{T1=b}
{P=ac∘P' T1=C}
% P2
edge (a,b)
path (b,b,P')
False
Path (C,b,P')
% P3
True
P=[a,c,b]
edge (C,b)
edge (C,T)
path (T,b,P')
{T2=b}
{T2=a} P'=ca∘P''
edge(c,b)
path (b,b,P'')
edge(c,a)
path (a,b,P'')
% P3
% P4
False
Path (b,b,P'')
Path (a,b,P'')
% P1
True
P=[a,c,a,b]
edge (a,b)
edge (a,T3)
path (T3,b,P''')
לולאה אין סופית...