***PPL Assignment 5***

Part 1

a.

Define an equivalence criterion for two lazy lists (when do we say that two lazy lists are equivalent)

Given the following two lazy lists lazy1 and lazy2 there are equivalent if and only if :

Both are empty – lazy1 = lazy2 = ‘()

Or

For each n the value returned by the calculation of the n’th function is the same for both lists.

They could be either finite or infinite.

For clarification – for any n – number of using tail

head (tail (tail (…tail (tail lazy1))…))) = head (tail (tail (…tail (tail lazy2))…)))

b.

In this case both of the lazy lists are infinite – non empty.

So we must proof they satisfy the last part of the definition of our equivalent definition.

For any n the following is satisfied :

head (tail (tail (…tail (tail lazy1))…))) = head (tail (tail (…tail (tail lazy2))…)))

lema –

even-square-1 - first the function g = (lzl-map (lambda (x) (\* x x)) is activated creating a list of all natural numbers squared. Then this list will be filtered with (lzl-filter (lambda (x) (= (modulo x 2) 0)) - we will call it f leaving only the even numbers. Using the lema we conclude that each element in the final list is removed if and only if . The functions f,g are commutative so the result will be the same as for even-square-1 which uses the functions f,g in the other order.

Now we will show that – for the given n

head (tail (tail (…tail (tail lazy1))…))) =

g(integers-from 0) =>

f(g(integers-from 0)) => - making the ’th value of the list.

And the following will conclude the question.

head (tail (tail (…tail (tail lazy2))…))) =

f(integers-from 0) => - making the ’th value of this list.

f(g(integers-from 0)) =>

for summery – for any n representing the number of times we use tail as described above. The value received from head is the ’th element of both of the lists and it is the same, it returns -

Part 2

1. Let’s define the equivalence between a function foo == foo$ and foo$ as it’s success-fail-continuations to be as followed:

foo is equivalent to foo$ if and only if –

foo(a) => ‘fail ⬄ foo$(a) => fail

foo(a) => suc : T ⬄ foo$(a) => success(suc : T)

Part 3

3.1

a. The unification will fail.

Unify [ t ( s ( s ) , G , H , p, t ( E ) , s ) ,

t ( s ( H ) , G , p , p, t ( E ) , K ) ]

sub = {}

2 composite terms that are atomic formulas with predicate t.

Both have same number of arguments so well compare each 2.

Equations: s(s)=s(H), G=G, H=p, p=p, t(E)=t(E), s=K

s(s)=s(H) same comparison so well compare the insides -> sub = {H=s}

G=G -> skip, H=p -> apply sub well get s=p => failure.

b. The unification will fail.

Unify [ g ( c , v ( U ) , g , G , U , E , v ( M ) ) ,

g ( c , M , g ,v ( M ) , v ( G ) , g , v ( M ) ]

sub = {}

2 composite terms that are atomic formulas with predicate g.

Both have same number of arguments so well compare each 2.

Equations: c = c, v(U) = M, g = g, G = v(M), U = v(G), E = g, v(M) = v(M)

c=c -> skip, M=v(U) -> sub={ M=v(U) }, g=g -> skip

G=v(M) -> apply sub and add -> sub={ m=v(U), G=v(v(U)) }

U=v(G) -> apply sub and fail -> U=v(v(v(U))) => same parameter on both sides is a failure.

c. The unification will fail.

Unify [ s ( [ v | [ [ v | V ] | A ] ] ) ,

s ( [ v | [ v | A ] ] ) ]

sub = {}

2 composite terms that are atomic formulas with predicate s.

Both have same number of arguments so well compare each 2.

Equations: [ v | [ [ v | V ] | A ] ] = [ v | [ v | A ] ]

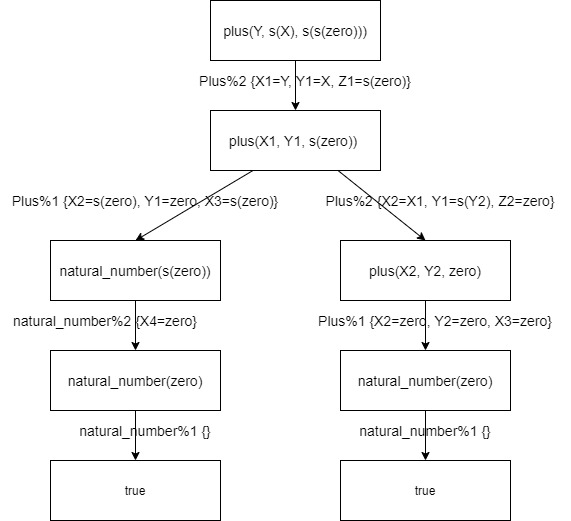
Unification between 2 pairs so well compare each 2 same index.

v=v -> skip, [ [ v | V ] | A ] = [ v | A ] -> another pairs, same comparison

v=[ v | V ] -> fail, we got symbol on one side and composite argument on other, A=A -> skip

3.1

a. proof tree? - plus(Y , s(X) , s(s(zero)))



Left:

{} o {X4=zero} o { X2=s(zero), Y1=zero, X3=s(zero)} o {X1=Y, Y1=X, Z1=s(zero)} =

{X4=zero, X3=s(zero), Y1=zero, X1=s(zero), Y=s(zero), X=zero, Z1=s(zero)}

Answer: {Y=s(zero), X=zero}

Right:

{} o {X2=zero, Y2=zero, X3=zero} o {X2=X1, Y1=s(Y2), Z2=zero} o X1=Y, Y1=X, Z1=s(zero)} =

{X2=zero, Y2=zero, X3=zero, X1=zero, Y1=s(zero), Z2=zero, Y=zero, X=s(zero), Z1=s(zero)}

Answer: {Y=zero, X=s(zero)}

b. The answer is: { {Y=s(zero), X=zero}, {Y=zero, X=s(zero)} }

c. This is a success proof tree because it has at least 1 successful path.

d. This tree doesn’t have infinite computational path, so this tree is finite