1. טענה: הפרוצדורה append$ שקולה-CPS לפרוצדורה append. כלומר לכל שתי רשימות ופונקציית ההמשך cont יתקיים (append$ lst1 lst2 cont) = (cont (append lst1 lst2))
נוכיח את הטענה באינדוקציה על אורך הרשימה lst1:
נסמן |lst1| = n
<u>בסיס האינדוקציה:</u> n = 0

a-e[append$ ‘() lst2 cont] = a-e[(cont (append ‘() lst2))]
⇒
a-e[cont lst2] = a-e[(cont lst2)]

<u>הנחת האינדוקציה:</u> עבור כל $n = k \in \mathbb{N}$ הטענה מתקיימת לכל $|lst1| = i \leq k$ כלומר:
(append$ lst1 lst2 cont) = (cont (append lst1 lst2))

<u>צעד האינדוקציה:</u> יהי $k \in \mathbb{N}$, $|lst1| = n = k + 1$ אזי:

$$a - e[\ (append\$\ lst1\ lst2\ cont)] =>$$

$$a - e[\ (append\$\ (cdr\ lst1)\ (lst2) \left(lambda\ (appe - cdr)\left(cont\ (cons\ (car\ lst1)\ appe - cdr)\right)\right)]$$

מהנחת האינדוקציה נקבל:

$$a - e\ [\ \left(lambda\ (appe - cdr)\left(cont\ (cons\ (car\ lst1)\ appe - cdr)\right)\right) append\ (cdr - lst1) lst2))]$$

$$=>$$

$$a - e\left[\ cont\left(\ cons\ (car\ lst1)(append\ (cdr - lst1) lst2)\right)\right] =>$$

$$a - e\ [\ cont\ (append\ lst1\ lst2)]$$

2.d

<u>Reduce1-lzl:</u> we will use this function when we want to iterate over a complete lazy-list with a reducer and an initial value. This function will get in an infinite loop if the lazy-list isn’t finite.

<u>Reduce2-lzl:</u> same as 1, but will work on an infinite lazy-list because we have a limit (n).

<u>Reduce3-lzl:</u> same as 1 with the addition of saving all the “init” values we found during the iteration.

2.g

In the new function we implemented there’s no need for any parameters, this means we are able to generate a lazy list of an approximation of Pi, each generation like this take O(1) time to compute the value. In addition, because there are no parameters, the user that uses this function doesn't need any primal knowledge of Pi computation methods to get the approximation, in contrast to Pi-Sum taught in class.

The disadvantage of the new function is that it returns a value as a lazy-list and not the directly the number like Pi-Sum.

3.1

a. Unify(A,B)

A = { t(s(s), G, s, p, t(K), s)}

B = { t(s(G), G, s, p, t(K), U)}

1. s= {s=G} A∘s= { t(s(s), s, s, p, t(K), s)}

B∘s= { t(s(s), s, s, p, t(K), U)}

2. s={s=U, G=U} A∘s= { t(s(s), G, s, p, t(K), s)}

B∘s= { t(s(G), G, s, p, t(K), s)}

s is now the most general unifier.

b. Unify(A,B)

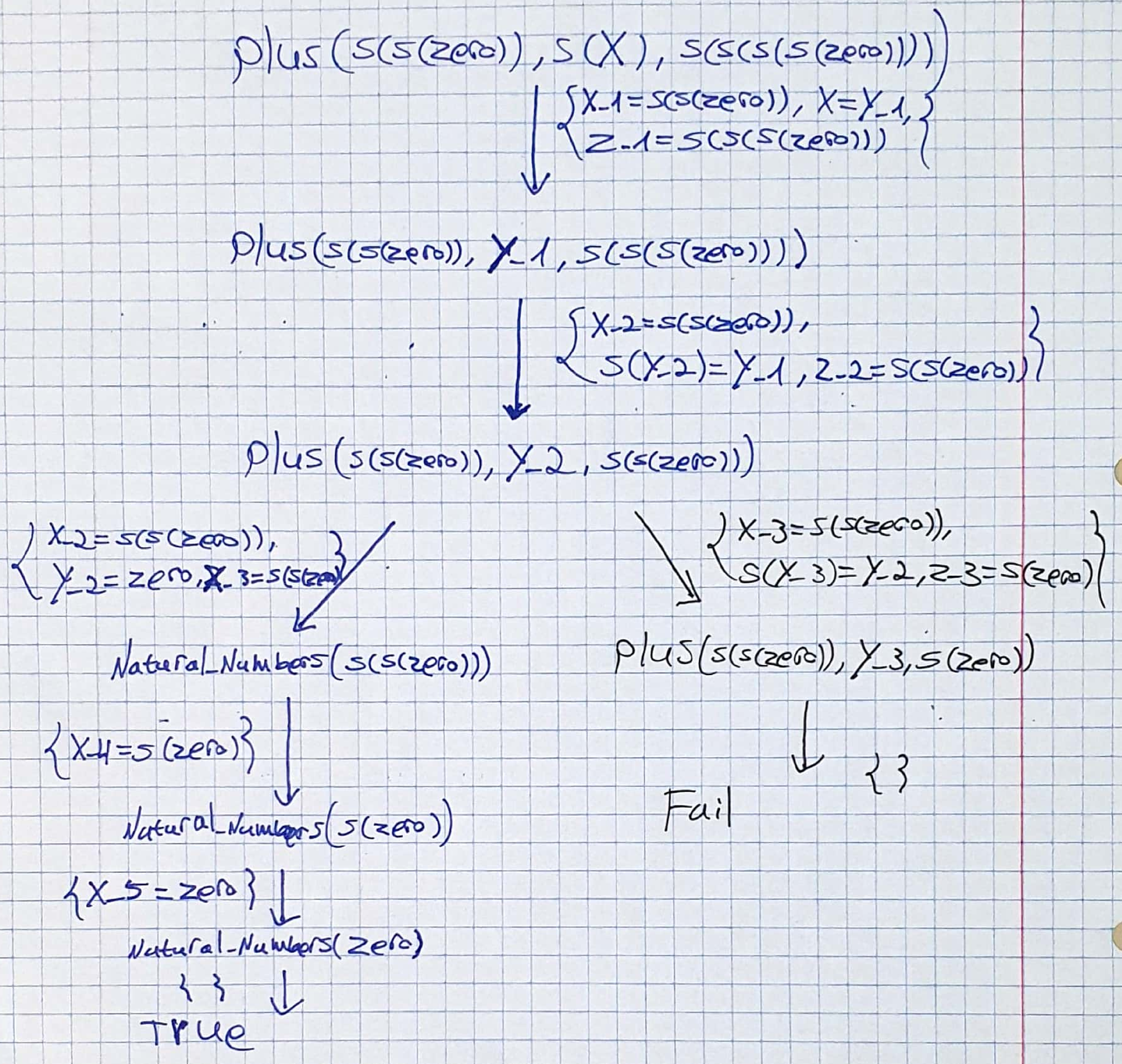
A = { p([v | [V | W]]) }

B = { p([[v | V] | W]) }

1. s= {v= [v | V] }

We get the s unifier but while running occurs check we get **Fail** because there is circularity (just like x=f(x) we saw in class).

3.3



Left Path:

⟹ s={}

⟹ {X_5=zero}

⟹ {X_4=s(zero)}

⟹ {X_2=s(s(zero)), Y_2=zero, X_3=s(s(zero))}

⟹ {X_2=s(s(zero)), [illegible] Y_1=s(zero), Z_2=s(s(zero))}

⟹ {X_1=s(s(zero)), X=s(zero), Z_1=s(s(s(zero)))}

▨ X=s(zero) אכן קיבלנו ⟸