שאלה 2.a

1. Two given generators / lazy lists , l1 and l2, are equivalent if and only if
For every $i \in N$:

   (*) Generators - $take(l1, i) = take(l2, i)$

   (**) Lazy lists – (eq? (take l1 i) (take l2 i)) ➔ #t

2. We will prove that evenSquares1 and evenSquares2 generators equivalent by induction on n, the number of elements to generate (the 2^{nd} parameter of take).

   Base case: let n=0. So take will return an empty list on every generator, particularly On evenSquares1, evenSquares2.

   Means, - $take(\text{evenSquares1}, 0) = take(\text{evenSquares2}, 0)$

   assumption: Assume, for *n* = *k*, that (*) holds for every *i, i* <= *k*; that is, that

   $take(\text{evenSquares1}, i) = take(\text{evenSquares2}, i)$

   Proof: let n=k+1. According to our assumption, the lists that we get from performing take in the lists are equal until their k element. k^{th} value is supposed to be an even number & a square number. Let's call the next number that holds both conditions m.

   EvenSquare1 - mapGen is a generator of all square numbers, starting from k^{th} value (we already returned k).
   FilterGen will return a generator of all square numbers that are even, starting from k^{th} value.
   When we perform take for the k+1 time on filterGen, it will return the next number on its list that holds both conditions (even & square) , means m.

   EvenSquare2 - FilterGen will return a generator of all even numbers, starting from k^{th} value.
   MapGen will return a generator of all even numbers that are square numbers, starting from k^{th} value.
   When we perform take for the k+1 time on mapGen, it will return the next number on its list that holds both conditions (even & square) , means m; that is, the k+1 element is equal in both results, and -

   $take(\text{evenSquares1}, k+1) = take(\text{evenSquares2}, k+1)$

3. We will prove that fibs1 and fibs2 lazy lists equivalent by induction on n, the number of elements to generate (the 2nd parameter of take).

Base case: let n=0. So take will return an empty list on every lazy list, particularly On fibs1, fibs2.

Means, - (eq? (take fibs1 0) (take fibs2 0)) ➔ #t

assumption: Assume, for *n* = *k*, that (**) holds for every *i, i* <= *k*; that is, that

(eq? (take fibs1 i) (take fibs2 i)) ➔ #t

Proof: let n=k+1. According to our assumption, the lists that we get from performing take in the lists are equal until their k element. k^{th} value is supposed to be next number on Fibonacci squence. Let's call it m.

every time we call take function, it gets the next element of the lzl (head lz-lst) and performs the lambda of the lzl. So according to the assumption, the k-1 and k elements are valid.

fibs1 – the lambda of the list is a recursive call, that creates the next value by adding the k-1 and k elements, so it is the next number on Fibonacci squence – m.

fibs2 – it is built like - (cons a (cons b(cons c …)))…)
the lambda of the list is a call to lz-lst-add , which receives 2 lazylists and creates the next cons of the lzl, by adding the head of the 2 lists. We call lz-lst-add with the k and k-1 elements, so we get the next number on Fibonacci squence – m; that is, the k+1 element is equal in both results, and -

(eq? (take fibs1 [k + 1]) (take fibs2 [k + 1])) ➔ #t

שאלה 3.a.2

נוכיח את הטענה באינדוקציה על n גודל הרשימה הראשונה.

בסיס: n=0 –

a-e[(append$ '() l2 c)] ➔* a-e[(c '() l2)] = a-e[(c (append '() l2))]

הנחה: עבור n=k€N הטענה מתקיימת לכל i, k>=i . כלומר, אם גודל l1 הוא i, אז מתקיים :

(append$ l1 l2 c) = (c (append l1 l2))

צעד: יהי k€N, n=k+1 , כלומר, אם גודל l1 הוא n=k+1 אז:

a-e[(append$ l1 l2 c)] ➔*

נכנס למקרה ה-else, ולכן יתבצע :

a-e[(append$ (cdr l1) l2 (lambda(res-acc) (c (cons (car l1) res-acc))))]

➔*

נשים לב שהרשימה (cdr l1) שנשלחת ל-append$ היא בגודל k, ולכן מהנחת האינדוקציה, נקבל ש-

a-e [((lambda(res-acc) (c (cons (car l1) res-acc))) (append (cdr l1) l2))] ➔*

a-e [(c (cons (car l1) (append (cdr l1) l2)))] =

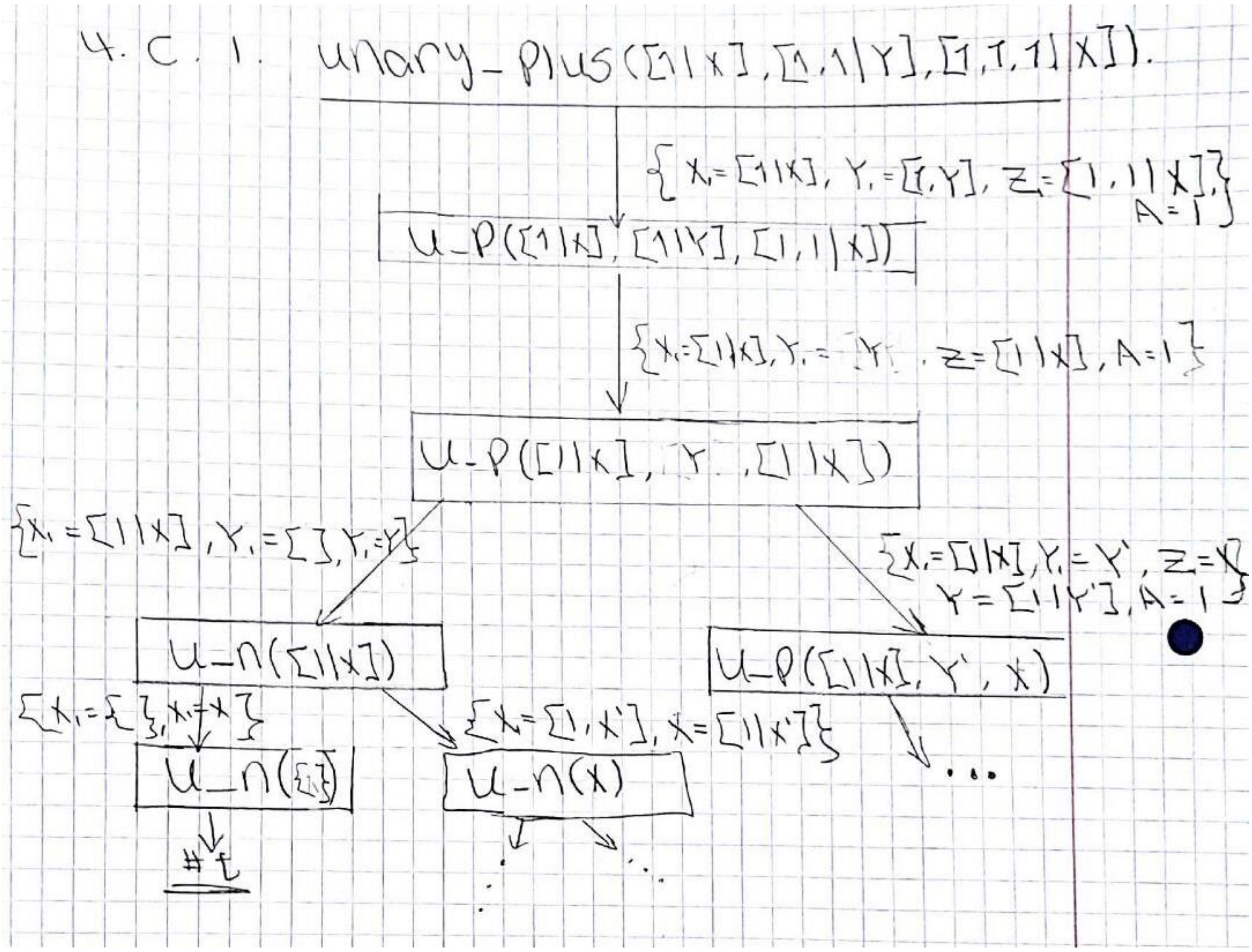
a-e [(c (append l1 l2))]

שאלה 4

4.b. The result of all these operations is "fail" because the number of arguments inside is unequal on both side.
The prosses of the algo unify:

> Open the proc p for both side, after the proc v, and in that moment, there are not the same number of arguments.

4.c. [1]



[2] Is a success, because the one of the leaf return true.

[3] The tree is infinite because the tree can calcitonin an infinite set of value that answer true.

Example: X = Y = [1 | []], [1 | 1 | []] ...