Assignment 4

2.1

Criterions for equivalence of a lazy list or generator :

The first item in both of the lists will be equal and the functions in the second item in the lists will be generator functions that are equivalent functions from the definition of functional equivalent

2.2

Const evensquares1=filterGen(mapGen(naturalNumber(), x=>x*x), x=>(x%2===0);

Const evensquares2= mapGen(filterGen(naturalNumber(), x=>x*x),
x=>(x%2===0);

Criterions for equivalence of a lazy list or generator :

The first item in both of the lists will be equal and the functions in the second item in the lists will be generator functions that are equivalent functions from the definition of functional equivalent .

We will prove that evenSqure1 equivalent to evenSquare2 by induction on recursive calls.

Base: n=1: for evenSqure1 we will compute the map on the function x*x and we will get 0 , the filter function return true and evenSqure1 return 0.

For evenSqure2 we will compute the filter function on 0 and return true than the map on the function x*x and we will get 0 and evenSqure2 return 0.

Induction step : assume for n and prove for n+1 steps:

For the first n steps ,from the induction assumption evensqure1=evensqure2 .

For the n+1 step ,

For the function evenSqure1 we will compute the square of the number and then we will filter the odd numbers ,a square of a number keep its parity the same and the filter will drop the odd numbers.

the computation of evenSqure2 is on the even number (if its take odd number the function of the filter will drop it) and after filter the odd numbers it will compute square on the even number.

We can see that we gets the same value in the first n items and we get the same computation on the n+1 item.

So the function are equivalent.

2.3

We will prove that fib1 equal to fib2 by induction on the recursive steps.

Base: n=0:

For fib1: we will call to fibGen with the values 0 and 1 and will build a lazy list that the first element is 0 and the second element is procedure of the sum of 1 and 0;

For fib2: we will build a lazy that here first element is 0 and the second is a procedure.

Base n=1:

For fib1: we will apply the procedure on the list from the previous step and get a lazy list that the first item is the sum of 0 and 1 and that the second item in Fibonacci series ,and the second item in the list is a procedure that call fibGen with the current item and the sum of the current item and the previous item.

For fib: we will apply the procedure on the list from the previous step and get a lazy list that has the value 1 and the second item is procedure that call lz-lst-add that get 2 lazy list that one of them is fib2 and the other one is the tail of fib2 and compute the sum of the two heads of the lists and return a lazy list that here first part of it is the sum of the heads and the other one is function that call recursively to lz-lst-add with the tail of both lists.

Induction assumption: assume the correctness on n steps and prove on n+1 steps.

For fib1:for the n+1 item in Fibonacci series ,for the n steps from the induction assumption we will get the n number in Fibonacci series and for the n+1 item we will sum the value on the n place in the list and the n-1 and get the n+1 number in Fibonacci series that process is the same to base case 1.

For fib2:for the n+1 item in Fibonacci series ,for the n steps from the induction assumption we will get the n number in Fibonacci series and for the n+1 item we will pay attention that in order to calculate the n number and the n-1 number is Fibonacci series from the induction assumption we will get two lazy list with that the first item is n and n-1 items in Fibonacci series and when we send those lists to the function lz-lst-add we will get a lazy list that the first item is the sum of n and n-1 and we will get the n+1 item is Fibonacci series and the second item is a procedure that get the tail that call lz-lst-add with the tail of the two list .

For conclusion both function return the same values.

3.2

נוכיח כי (append$ lst1 lst2 cont)=(cont (append lst1 lst2)).

נוכיח באינדוקציה על אורך הרשימה - lst1. נסמן אורך זה כך: |lst1|

בסיס:

|lst1|=0 כלומר מדובר ברשימה ריקה לכן:

AE[(append$ lst1 lst2 cont)] ==>

AE[(if (null? lst1') (cont lst2')

(append$ (cdr lst1') lst2'

(lambda (res) (cont (cons (car lst1') res)))))] ==>

If is special form – AE[(null? lst1')] = #t

Therefore :

AE[(append$ lst1 lst2 cont)] = AE[(cont lst2')]

AE[(cont (append lst1 lst2))] ==>

```
AE[(append lst1 lst2)] =AE[(if (null? lst1')
                                 lst2'
                                (cons (car lst1')
                                      (append (cdr lst1') lst2')))]
```

= if is special form – AE[(null? lst1)] = #t

Therefore : AE[(append lst1 lst2)] = AE[lst2'] =lst2'

AE[(cont (append lst1 lst2))] = AE[(cont lst2')]

Therefore: AE[(cont (append lst1 lst2))] = AE[(cont lst2')] = AE[(append$ lst1 lst2 cont)]

הוכחנו את מקרה הבסיס . נניח נכונות ל n=>0 |lst1| ונוכיח ל |Lst|=n+1

AE[(append$ lst1 lst2 cont)] ==>

AE[(if (null? lst1') (cont lst2')

(append$ (cdr lst1') lst2'

(lambda (res) (cont (cons (car lst1') res)))))] ==>

If is special form – AE[(null? lst1')] = #f

Therefore :

AE[(append$ lst1 lst2 cont)] = AE[(append$ (cdr lst1') lst2'

(lambda (res) (cont (cons (car lst1') res))))]

ומהנחת האינדוקציה מתקיימת-

AE[(append$ (cdr lst1') lst2'

(lambda (res) (cont (cons (car lst1') res))))] =

AE[((lambda (res) (cont (cons (car lst1') res))) (append (cdr lst1') lst2'))] =

AE[(cont (cons (car lst1') (append (cdr lst1') lst2')))) =

AE[(cont (append lst1' lst2'))]

אזי מתקיים: (append$ lst1 lst2 cont)=(cont (append lst1 lst2)).

4b

1)

. unify[p(v(v(d(1), M, ntuf3), X)), p(v(d(B), v(B, ntuf3),

KtM))] Sub={} p=p so we unify inner AtomicFormula

v=v so we unify the inner AtomicFormula

the length of parameters of AtomicFormulas isn’t equal so unify faill.

2)

b. unify[p(v(v(d(1), M, ntuf3), X)), p(v(d(B), v(B, ntuf3), ntuf3))]

Sub={} p=p so we unify inner AtomicFormula

v=v so we unify the inner AtomicFormula

the length of parameters of AtomicFormulas isn’t equal so unify faill.

3)

c. unify[p(v(v(d(M), M, ntuf3), X)), p(v(d(B), v(B, ntuf3), KtM))]

Sub={} p=p so we unify inner AtomicFormula

v=v so we unify the inner AtomicFormula

the length of parameters of AtomicFormulas isn't equal so unify faill.

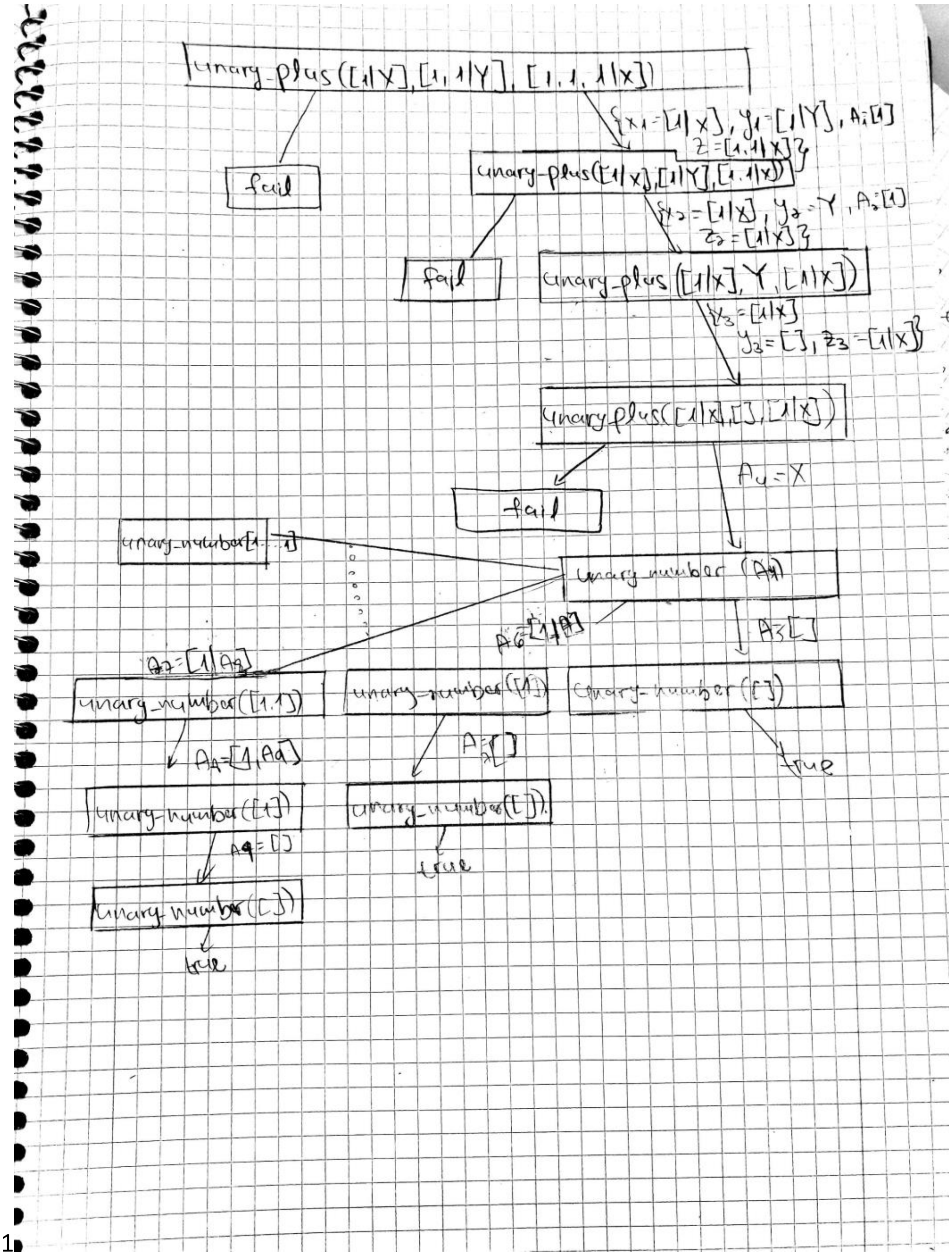
4)

unify[p(v(v(d(1M, p), X)), p(v(d(B), v(B, ntuf3), KtM))]

Sub={} p=p so we unify inner AtomicFormula

v=v so we unify the inner AtomicFormula

the length of parameters of AtomicFormulas isn't equal so unify faill.



c.1

c.2.Yes, this is a success tree because there are nodes that return true.

c.3.yes, we can see that there are infinity substitutions for x at the node unary_number(A)

That will take us to true node.