Question 1:

1.1:

a. Let list1, list2 be two lazy lists.

We say that $list1 = list2$ if and only if:

For every $n \in N$:

$(nth\ list1\ n) = (nth\ list2\ n)$

where "nth" is the following function (shown in class):

```
(define nth
        (lambda (lz-lst n)
                (if (= n 0)
                  (head lz-lst)
                     (nth (tail lz-lst) (sub1 n)))))
```

** every function nth' that gets the nth value from a lazy list is valid for the definition.

b.

$Proof\ by\ induction\ on\ n$:

Base case n=0:

$(nth\ \mathrm{even-square-1}\ 0) = (nth\ \mathrm{even-square-2}\ 0) = 0$

this indicates that base case is valid.

lets assume that the claim is true for k<n and prove on n:

$(nth\ \mathrm{even-square-1}\ n) =$ (lambda ($\mathrm{even-square-1}$ n)
(if (= n 0)
(head $\mathrm{even-square-1}$)
(nth (tail $\mathrm{even-square-1}$) (sub1 n)))))

= (lambda ($\mathrm{even-square-1}$ n)
(if (= n 0) **: [base case of induction, switch $\mathbf{even-square-1}$by $\mathbf{even-square-2}$**
(head $\mathrm{even-square-2}$)
(nth (tail $\mathrm{even-square-1}$) n-1)))) **: Apply sub1 on n**

Observation: if for 2 lazy lists lzl1 , lzl2 $(nth\ lzl1\ n) = (nth\ lzl2\ n)$ for every n , then $(nth\ (tail\ lzl1)\ n) = (nth\ (tail\ lzl2)\ n)$ that’s because for every lazy list lzl:

$(tail\ lzl) \subseteq lzl.$

Back to proof:

= (lambda $(\text{even} - \text{square} - 1$ n)

(if (= n 0)

(head $\text{even} - \text{square} - 2)$

(nth (tail $\text{even} - \text{square} - 2)$ n-1)))) **: observation + induction**

= $(nth\,\text{even} - \text{square} - 2\ n)$ **: By definition.**

Question 2:

a.

A procedure $f{:}\,[x_1 * \ldots * x_n \rightarrow T_1 \cup T_2]$ is equivalent to it's success-fail continuation version $f\$$: $[x_1 * \ldots * x_n * [T_1 \rightarrow S_1] * [Empty \rightarrow S_2] \rightarrow S_1 \cup S_2]$ if for every input $x_0, x_1, \ldots x_n$ and for every successFunc, failFunc:

1. if $f(x_0, \ldots x_n) \in T_1 =>$
$$f\$(x_0, \ldots x_n,\ success, fail) = succsess(f(x_0, \ldots x_n))$$

2. if $f(x_0, \ldots x_n) \in T_2 =>$
$$f\$(x_0, \ldots x_n,\ success, fail) \in S_2$$

* Assuming (without the lost of generality) that type T_1 is a success scenario of procedure f and T_2 is a fail scenario of procedure f.

d.

Claim: the procedure get-value$ is success-fail continuation version of procedure f.
Then, for every association list "lst", key "key", successFunc "success", failFunc "fail":

1. if $get - value(lst, key) \in \mathrm{T} =>$
$$get - value\$(lst, key, \quad success, fail) = succsess(get - value(lst, key))$$

2. if $get - value(lst, key) \in \{'fail\} =>$
$$get - value\$(lst, key,\ success, fail) \in T_2$$

Proof:

let "lst" be an association list, "key" key, "success" success continuation and "fail" fail continuation.

We divide the proof in two cases:

Case 1: assume $get - value(lst,key) \in \mathrm{T}$, need to proof that:

$get - value\$(lst, key,\ success, fail) = succsess\,(get - value(lst, key))$

$get - value(lst, key) \in \mathrm{T}$, so by definition a success scenario is performed in

"get-value". So there exists a pair <key,x> that "get-value" return the value of the pair - x. In addition, get-value$ would find the pair <key,x> in the association list (because we know the pair is exists) and apply the success continuation "success" on the value of the pair when get-value$ find the value of the pair and return.

So get-value$ return: $success\,(x)$, and it is the same as finding x in get-value, and then perform the success procedure on x. =>

$$get - value\$(lst, key, success, fail) = succsess\,(get - value(lst, key))$$

Case 2: assume $get - value(lst,key) \in \{'fail\}$ we need to proof that:

$$get - value\$(lst, key, success, fail) \in T_2$$

assume $get - value(lst, key) \in \{'fail\}$ so we can conclude thar get-value returned 'fail. That means that there is no pair that it's key is "key". So we can conclude that get-value$ perform and return it's fail continuation "fail". It's a function with the type: $Empty \to T_2$ so $get - value\$(lst, key, success, fail) \in T_2$.

We took care of both cases, and that ends the proof.

Question 3:

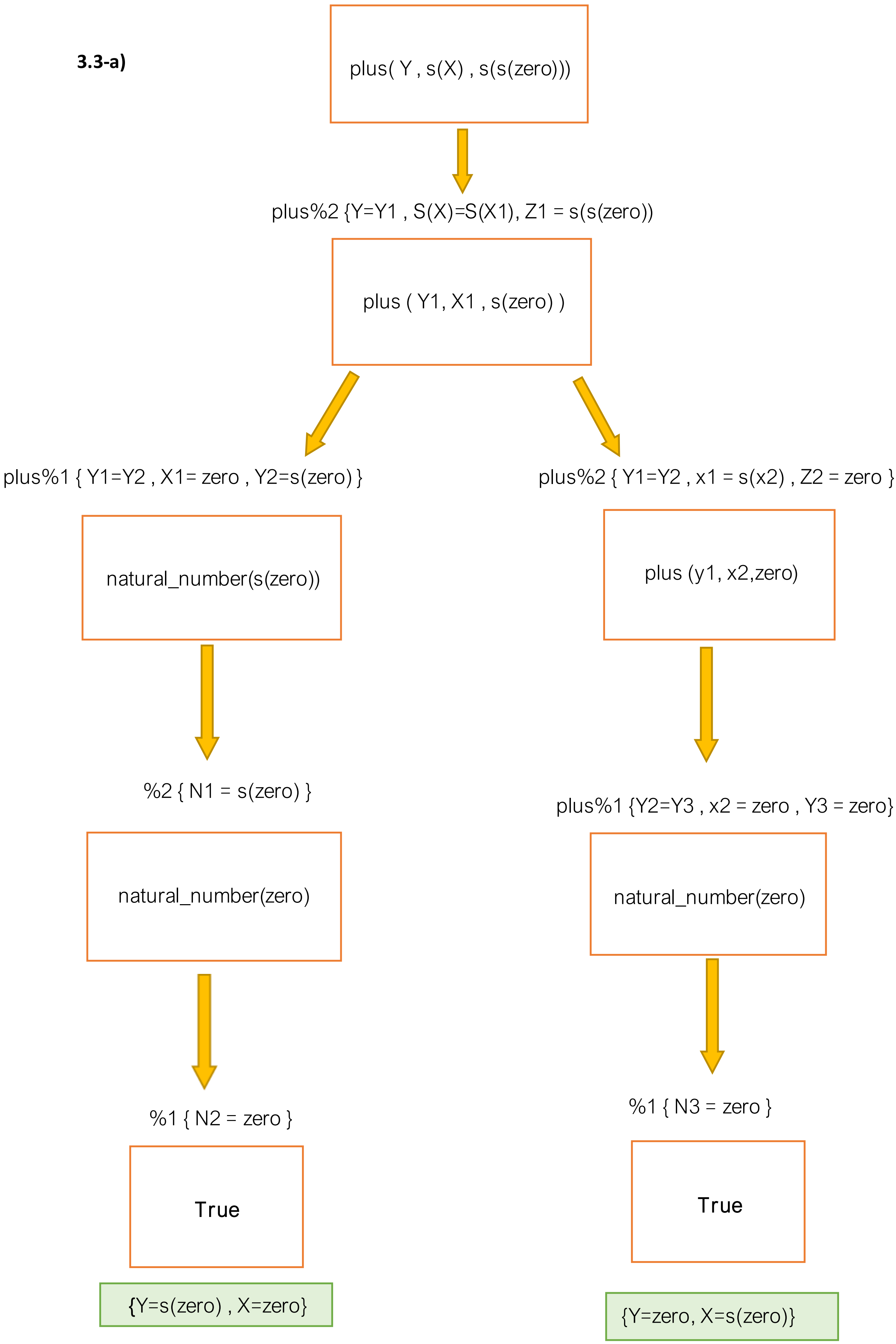
3.3-b) the answers are :
X = zero , Y = s(zero)
X = s(zero) , Y = zero
false.

3.3-c) success – because there is a path that leads to success leaf.
3.3-d) finite – because all the paths are finite.

3.1)

- Unify [t (s (s) , G , H , p, t (E) , s) ,
  t (s (H) , G , p , p, t (E) , K)]
  { s = H , H = p , s = K } – failure
- Unify [g (c , v (U) , g , G , U , E , v (M)) ,
  g (c , M , g ,v (M) , v (G) , g , v (M)]
  { U = v(v(v(U))) , E = g } – failure
- Unify [s ([v | [[v | V] | A]]) ,
  s ([v | [v | A]])]
  { [v | V] = v } - failure

3.3-a)


plus(Y , s(X) , s(s(zero)))
plus%2 {Y=Y1 , S(X)=S(X1), Z1 = s(s(zero))
plus (Y1, X1 , s(zero))
plus%1 { Y1=Y2 , X1= zero , Y2=s(zero) }
plus%2 { Y1=Y2 , x1 = s(x2) , Z2 = zero }
natural_number(s(zero))
plus (y1, x2,zero)
%2 { N1 = s(zero) }
plus%1 {Y2=Y3 , x2 = zero , Y3 = zero}
natural_number(zero)
natural_number(zero)
%1 { N2 = zero }
%1 { N3 = zero }
True
True
{Y=s(zero) , X=zero}
{Y=zero, X=s(zero)}