Lazy List:

Q1- a-

Lazy list is defined by : LZl = EmptyLZl | Pair(T,Empty->LZl(T))

Two lazy lists : LZl1 ,LZl2 , are equivalent if:

1) if LZl1 is EmptyLZl => (LZl2 = LZl2 ) <=> LZl2 = EmptyLZl and vice versa

2) else:

LZl1 = Pair(x, Empty->L3)

LZl2 = Pair(y, Empty->L4)

than,

(LZl1 = LZl2) <=> (x=y) & ((Empty->L3 = Empty->L4 <=> L3 = L4))

Q1- b-

(define even-square-1

(lzl-filter (lambda (x) (= (modulo x 2) 0))

(lzl-map (lambda (x) (\* x x))

(integers-from 0))))

(define even-square-2

(lzl-map (lambda (x) (\* x x))

(lzl-filter (lambda (x) (= (modulo x 2) 0))

(integers-from 0))))

By the definition above even-square1 == even-square-2 if:

Lets mark : even-square-1 = Pair(x, Empty->Lzl1)

even-square-2 = Pair(y, Empty->Lzl2)

1) We'll show x=y:

(integers-from 0) returns a list (0, integers-from 1)

therefor, the computation ((x) (\* x x)) on 0 will result in 0,

0%2 = 0, and therefor stands in the presicate and will be included in the final list => x=0

(integers-from 0) returns a list (0, integers-from 1)

therefor, the computation x%2 on 0 will result in 0,

and after lzl-map we'll get that the first item in the final list is

0 since 0^2=0 => y=0

=> x=y

2) We'll show Empty->Lzl1 = Empty->Lzl2:

in even-square-1:

on each item x we'll get after calling next from (integers-from

x+1) ,

lzl-map will compute its square x^2 and after lzl-filter computation,the value

x^2 will return.

if and only if x^2%2 = 0, else, will compute the next value of lzl-map.

therefor the next item that return will be the square of the next even number in

(integers-from x+1)

in even-square-2:

on each item y we'll get after calling next from (integers-from y+1) ,

lzl-filter will return y as the next value only if y is even (else will compute the next

value from integers-from), after lzl-map computation,the value y^2 will return as the

next value.

therefor the next item that return will be the square of the next even number in

(integers-from y+1), we have already shown x=y => square of next even number in

(integers-from y+1) and (integers-from x+1) are identical.

Q2- a-

For f:[x1\*..\*xn -> xt] and f$:[y1\*..\*yn \* succ-cont \* fail-cont -> yt], a   
Success-Fail-Continuation version of it, we will say f=f$ iff:

For all x1,..,xn, and all succ,fail

(f$ x1 … xn succ fail) = (succ (f x1 … xn)) or (f$ x1 … xn succ fail) = (fail (f x1 … xn))

d-

Let assoc-list, key\_x and succ-cont, fail-cont.

If key is in assoc-list:

There is pair in assoc-list <key\_x,val\_x>, and

(get-value assoc-list key\_x) = val\_x ⬄

(succ-cont (get-value assoc-list key\_x)) = (succ-cont val\_x)

And (get-value$ assoc-list key\_x succ-cont fail-cont) = (succ-cont val\_x)

So –

(succ-cont (get-value assoc-list key\_x)) =

(get-value$ assoc-list key\_x succ-cont fail-cont).

If key is not in assoc-list:

(get-value assoc-list key\_x) will fail so the output will be

(fail-cont (get-value assoc-list key\_x)), and get-value$ will fail, because it’s the

same list, so fail-cont will be activated so –

(fail-cont (get-value assoc-list key\_x)) =

(get-value$ assoc-list key\_x succ-cont fail-cont).

-Q3.1

a. Unify [t(s(s), G, H, p, t(E), s) , t(s(H), G, p, p, t(E), K)] :

|  |  |
| --- | --- |
| Equations | Sub |
| ~~t(s(s), G, H, p, t(E), s) =~~  ~~t(s(H), G, p, p, t(E), K)~~ | {} |
|  |
|  |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~s(s) = s(H)~~ | {} |
| G=G |
| H = p |
| p = p |
| t(E)=t(E) |
| s=K |

|  |  |
| --- | --- |
| Equations | Sub |
| G=G | {} |
| H = p |
| p = p |
| t(E)=t(E) |
| s=K |
| s=H |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~G=G~~ | {} |
| H = p |
| p = p |
| t(E)=t(E) |
| s=K |
| s=H |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~H = p~~ | { H = p } |
| p = p |
| t(E)=t(E) |
| s=K |
| s=H |
|  |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~p = p~~ | { H = p } |
| t(E)=t(E) |
| s=K |
| s=H |
|  |
|  |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~t(E)=t(E)~~ | { H = p } |
| s=K |
| s=H |
| E=E |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~s=K~~ | { H = p, K = s } |
| s=H |
| E=E |
|  |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~s=H~~ | { s = p, K = s, s=H} |
| E=E |

s=p is two different constant symbols, algorithm FAIL.

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

|  |  |
| --- | --- |
| Equations | Sub |
| ~~g(c, v(U), g, G, U, E, v(M))=~~  ~~g(c, M, g, v(M), v(G), g, v(M))~~ | {} |
|  |
|  |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~c=c~~ | {} |
| v(U)=M |
| G=v(M) |
| U=v(G) |
| E=g |
| v(M)=v(M) |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~v(U)=M~~ | {M=v(U)} |
| G=v(M) |
| U=v(G) |
| E=g |
| v(M)=v(M) |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~G=v(M)~~ | {M=v(U),G=v(v(U))} |
| U=v(G) |
| E=g |
| v(M)=v(M) |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~U=v(G)~~ | {M=v(U),G=v(v(U)),U=v(v(v(U)))} |
| E=g |
| v(M)=v(M) |

U=v(v(v(U))) is fail due to the Occurs-check, recursive definition of U.

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

|  |  |
| --- | --- |
| Equations | Sub |
| ~~s([v|[[v|V]|A]]) = s([v|[v|A]])~~ | {} |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~[v|[[v|V]|A]] = [v|[v|A]]~~ | {} |

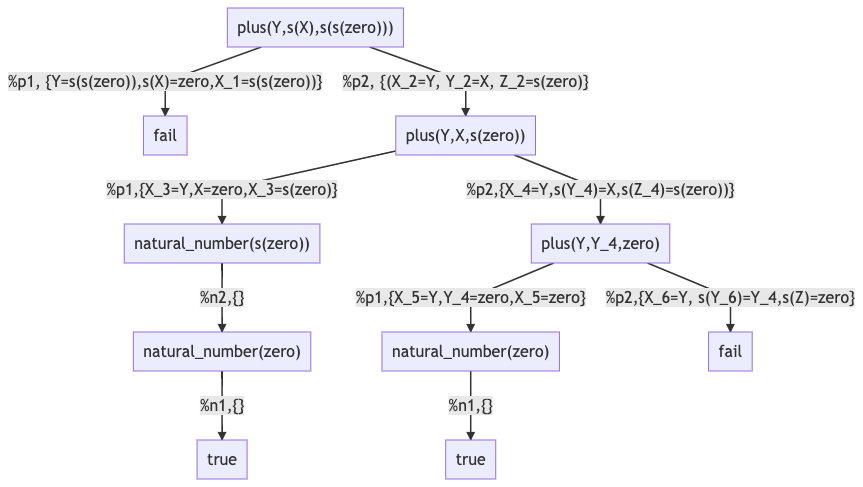
|  |  |
| --- | --- |
| Equations | Sub |
| ~~v= v~~ | {} |
| [[v|V]|A] = [v|A] |  |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~[[v|V]|A] = [v|A]~~ | {} |

|  |  |
| --- | --- |
| Equations | Sub |
| ~~[v|V] = v~~ | {} |
| A=A |  |

v=[v|V] is fail due to the Occurs-check, recursive definition of v.

Q3.3

1. 



\*\* There is no assignment of X that will satisfies the equation ‘s(X)=zero’.



b- The answers for the query ?-plus(Y,s(X),s(s(zero))) is:

{<Y=zero, X=s(zero)>, <Y=s(zero), X=zero>}.

c- This is a success tree, because it has at least one path that leads to “true” node.

d- The tree is finite, all its paths are ending in a leaf, no infinite paths.