**Part1:**

**Q1.1 b.**

Proof by induction on list1’s length: signed as n for the proof

We’ll proof that : for every list1,list2 lists, envey continuation named cont, that functions append and append$, are CPS-equivalent.

**(append**$ **(list1 list2 cont)) = (cont(append (list1 list2)))**

**Basis:** n=0; ->list1 is empty

a-e[ append$ (‘() list2 cont) ] 🡺\* a-e[(cont (list2))]= a-e [cont(append ( ‘() l2))]

**Induction assumption:** let’s assume, that for , s.t list1 length is n, the equation is valid:

**(append**$ **(list1 list2 cont)) = (cont(append (list1 list2)))**

**Induction step:** we’ll prove that the equation is valid for n+1:

a-e[ (append$ (list1 list2 cont)) ] 🡺\*

a-e [ (append$ ( (cdr list1) list2 (lambda (append-res)

(cont (cons (car lst1) append-res))))]🡺\*

From the assumption of induction:

a-e [ (lambda (append-res)

(cont (cons (car lst1) append-res)) (append ((cdr list1) list2)))))]🡺\*

a-e [ (cont (cons (car lst1) (append ((cdr list1) list2)))]🡺\*

a-e[(cont( append (list1 list2)))] Q.E.D

Q2.d

**The reduce1-lzl:** is good for finite lazy-list computations.

**The reduce2-lzl:** is good for infinite lazy-list computations, but we know the index of last member we need to compute till.

**The reduce3-lzl:** is good for approximations

Q2.g

**Advantage of implementation via lazy-list:** Every time we generate the next computation, without opening new frames, so we avoid the hazzard of stack overflow and crashing the system.

**Disadvantage of implementation via lazy-list:** We cannot set the b member – which responsible for the precision of our approximation’s computation

**Part3:**

**Q 1.**

**1)unify[t(s(s), G, s, p, t(K), s), t(s(G), G, s, p, t(K), U)]**

**Define S={} – empty substitution**

**Build the equation A=B 🡺 (t(s(s), G, s, p, t(K), s))= ( t(s(G), G, s, p, t(K), U))**

**Let’s assume that A= (t(s(s), G, s, p, t(K), s)) ; B=( t(s(G), G, s, p, t(K), U))**

**Apply S on A,B:**

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

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

**(t(s(s), G, s, p, t(K), s))= ( t(s(G), G, s, p, t(K), U)) , according to step 7 of unification algorithm:**

**s(G)=s(s) -> G=s ;**

**Change the S: S=S\*{G=s}={}\*{G=s}={G=s}**

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

**B\*S=B\*{ G=s }=( t(s(s), G, s, p, t(K), U)) , according to step 7 of unification algorithm:**

**G=G; continue (step 4)**

**s=s ; continue (step 4)**

**p=p ; continue (step 4)**

**t(K)=t(K) ; continue (step 4)**

**U=s -> Change the S: S=S\*{U=s}={G=s}\*{U=s}={G=s, U=s}**

**A\*S=A\*{ G=s, U=s }=(t(s(s), G, s, p, t(K), s))**

**B\*S=B\*{ G=s, U=s }=( t(s(s), G, s, p, t(K), s)) . so A=B, we have no more equations to solve, so the substitution is S={ G=s, U=s} is the mgu here.**

**2)** **unify[p([v | [V | W]]), p([[v | V] | W])]**

**Define S={} – empty substitution**

**Build the equation A=B 🡺 (p([v | [V | W]]))=( p([[v | V] | W])] )**

**Apply S on A,B:**

**A\*S= p([v | [V | W]])\*{}= p([v | [V | W]]) ;**

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

**(p([v | [V | W]])\*{}= (p([[v | V] | W])]) , according to step 7 of unification algorithm:**

**v = [ v ,V ]**

**As we can see we got LHS as a primitive(constant) , while the RHS is a compound type of list. So the equation will not satisfy any condition of the algorithm, and we have no mgu for current equation.**

**Q 3.**

**Diagram

Description automatically generated with medium confidence**