**Q1.1b**

Proposition: For any lists *lst1* and *lst2* and a continuation procedure *cont*, *(append$ lst1 lst2 cont) = (cont (append lst1 lst2))*.

Proof: By induction on the length of lst1

Base: For the case of a *lst1* of length *0* [the empty list], the value of *(append lst1 lst2)* is *lst2*, and the value of *(append$ lst1 lst2 cont)* is *(cont lst2)*, which implies *(append$ lst1 lst2 cont) = (cont (append lst1 lst2))*.

Induction step: We assume the proposition holds for *lst1* of length *n*, and show the proposition holds for *lst1* of a length *n+1*.

1. According to the code, the value of *(append lst1 lst2)* is *(cons (car lst1) (append (cdr lst1) lst2))*.
2. According to the code ,the value of *(append$ lst1 lst2 cont)* is *(append$ (cdr lst1) lst2 cont2)*, where *cont2* is the continuation procedure defined in lines 6-7.

Since the first operand of *(append$ (cdr lst1) lst2 cont2)* is a list of length *n*, according to the induction assumption: *(cont2 (append (cdr lst1) lst2)) = (append$ (cdr lst1) lst2 cont2)*.

* *(cont (cons (car lst1) (append (cdr lst1) lst2)))* = *(append$ (cdr lst1) lst2 cont2) ;;; code of cont 2*
* *(cont (append lst1 lst2)) = (append$ (cdr lst1) lst2 cont2) ;;; (a)*
* *(cont (append lst1 lst2)) = (append$ lst1 lst2 cont) ;;; (b)*

**Q2d**

reduce1-lzl: for a reduce of a finite lazy list

reduce2-lzl: for a reduce of one specific prefix of a given infinite lazy list

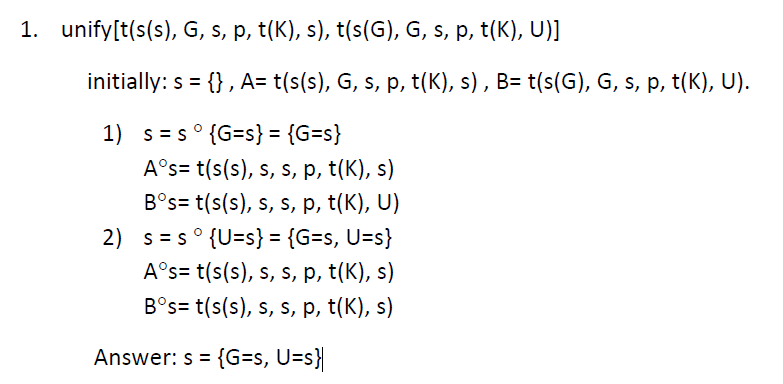
reduce3-lzl: for a reduce of each prefix of an infinite lazy list (as the case of Q2e)

**Q2g**

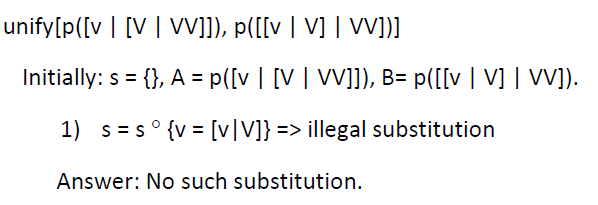
Advantage: can be applied for any approximation level, in contrast to pi-sum which is fixed to one given ‘b’ limit.

Disadvantage: generates a lot of closures.

**Q3a**



2.



**Q3c**

Note: In contrast to the figure bellow,

* No need to draw a fail-node in case there’s at least one selected rule.
* One fail node is sufficient in case no rule was selected.

