PPL – Assignment 5

Question 1

b. Proving append$ is CPS-equivalent to append:

We will prove: (append$ l1 l2 c) = (c (append l1 l2)).
We will prove by induction on the length of l1.

מקרה בסיס עבור 0 =|l1|:

a-e [(append$ l1 l2 c)] ➔ a-e[(c l2)] ➔ a-e[(c (append l1 l2))]
as needed.

הנחת האינדוקציה: נניח שעבור n =|l1| כש-n טבעי הטענה מתקיימת, כלומר:

(append$ l1 l2 c) = (c (append l1 l2))

צעד האינדוקציה: נוכיח שהטענה מתקיימת עבור n+1 =|l1|: (נסמן את הרשימה l1 באורך n ב- l1' לנוחות)

a-e[(append$ l1 l2 c)] ➔ a-e[(append$ (cdr l1) l2 **(**lambda (res) (c (cons (car l1) res))**)**]

נשתמש בהנחת האינדוקציה (אורך (cdr l1) הוא n ולכן ניתן להשתמש) ונקבל:

a-e[**(**(lambda (res) (c (cons (car l1) res))) (append l1' l2)**)**] ➔

➔ a-e[(c (cons (car l1) (append l1' l2)))] ➔

➔ a-e[(c (append l1 l2))]

מ.ש.ל

Question 2

d. We will use reduce1-lzl when we know the lists are finite, so we know the operation will end. We will use reduce2-lzl when we only want the operation applied to a finite number of elements in the lists, but the lists may be infinite. We will use reduce3-lzl when the lists are possibly infinite, and we want to apply the procedure a non-fixed number of times, possibly infinitely.

g. The advantage of generate-pi-approximations implementation comparing to pi-sum implementation is that we get to see every step in the approximation process, thus having a compact view of the recursive calls.

The disadvantage is that it is difficult to manipulate the outcome of the calculation, you have to build another method to simplify it, if needed.

Question 3.1

a. unify[t(s(s), G, s, p, t(K), s), t(s(G), G, s, p, t(K), U)]

s{}
A ○ s = t(s(s), G, s, p, t(K), s)
B ○ s = t(s(G), G, s, p, t(K), U)

s = s ○ { G = s } = { G = s}
A ○ s = t(s(s), s, s, p, t(K), s)
B ○ s = t(s(s), s, s, p, t(K), U)

s = s ○ {U = s} = { G = s, U = s}
A ○ s = t(s(s), s, s, p, t(K), s)
B ○ s = t(s(s), s, s, p, t(K), s)

Unification success → s = { G = s, U = s }

b. unify[p([v | [V | W]]), p([[v | V] | W])]

s={}

A ○ s = p([v | [V | W]])
B ○ s = p([[v | V] | W])

FAIL : v != [v | V] not the same structure.

Proof Tree is in the next page:

Proof Tree is in the next page:

Proof Tree is in the next page:

**For clarification reasons, I just mentioned the substitution but didn't always the variables in the text box for easier understanding of the process.