1.2) we wish to prove that append$ is cps-equivalent to append.

That is:

For any two lists and a a function “cont”
(append$ lst1 lst2 cont)=(cont (append lst1 lst2)).

We will prove the equation by induction over the length of lst1.

Base: n=0:

(append$ ‘() lst2 cont)=(cont lst2) (the predicate of the “if” statement holds.)

(Append ‘() lst2)=lst2, hence,(cont (append ‘() lst2)=(cont lst2),

Both sides equal (cont lst2) hence (append$ lst1 lst2 cont)=(cont(append lst1 lst2)).

Assumption:

We will assume that for any lst1 with length $k \leq n$,

(append$ lst1 lst2 cont)=(cont (append lst1 lst2))

Induction step:

Let lst1 be of size n+1.

(append$ lst1 lst2 cont) =

(append$ (cdr lst1) lst2 (lambda(res)(cont (cons (car lst1)res)))))

Now (cdr lst1) length is n, hence the assumption holds. Therefore:

(append$ (cdr lst1) lst2 (lambda(res)(cont (cons (car lst1)res)))))=

(lambda(res)(cont (cons (car lst1)res)))(append (cdr lst1) lst2)=

(cont (append lst1 lst2))

2.d We will use reduce1-lzl on finite lazy lists, otherwise we will get into an infinite loop (we can't reduce infinite number of elements). For example, calculate the sum of all the numbers in a finite lazy list.
We will use reduce2-lzl on infinite lazy lists, when we know the exact number of elements we want to reduce. For example, get the sum of the 10 first Fibonacci numbers.
We will use reduce3-lzl when we don't need the whole reduced value of the list immediately. For example, update an entry once an hour based on the reduced values of an infinite list. We want to access infinite number of elements, separately.

2.g The advantage of pi generator is that we can divide our calculations in a way which we can "save" old computations (getting the n+1 item from the lazy list requires the generation of one item and adding it to the previous sum, as opposed to calculating the whole approximation with pi-sum). This is helpful in situations where we don't know how accurate we need to be and might need to improve our calculations later on.
The disadvantage pf pi generator is that if we know how accurate we want to be, it will require multiple calls to the lazy list instead of calling pi-sum once. In that case pi-sum is more efficient.

3. 1 unify[t(s(s), G, s, p, t(K), s), t(s(G), G, s, p, t(K), U)].

Init: sub = {}, equations: [(t(s(s), G, s, p, t(K), s) = t(s(G), G, s, p, t(K), U)]

$$eq_1' = \big(\mathrm{t(s(s), G, s, p, t(K), s)} = \mathrm{t(s(G), G, s, p, t(K), U)}\big) \circ \{\,\}$$
$$= \big(\mathrm{t(s(s), G, s, p, t(K), s)} = \mathrm{t(s(G), G, s, p, t(K), U)}\big)$$

Both sides are atomic, same symbol predicate and same number of terms, so we split into more equations.

$$equations = [\big(s(s) = s(G)\big), (G = G), (s = s), (p = p), \big(t(K) = t(K)\big), (s = U)].$$

next iteration;

$$eq_1' = \big(s(s) = s(G)\big) \circ \{\,\} = \big(s(s) = s(G)\big)$$

Both sides are atomic, same symbol predicate and same number of terms, so we split into more equations.

$$equations = [(G = G), (s = s), (p = p), \big(t(K) = t(K)\big), (s = U), (s = G)].$$

next iteration;

$$eq_1' = (G = G) \circ \{\,\} = (G = G)$$

Same variable on both sides, we continue to the next iteration.

$$eq_1' = (s = s) \circ \{\,\} = (s = s)$$

same Constant symbol, continue to next iteration.

$$eq_1' = (p = p) \circ \{\,\} = (p = p)$$

same Constant symbol, continue to next iteration.

$$eq_1' = \big(t(K) = t(K)\big) \circ \{\,\} = \big(t(K) = t(K)\big)$$

Both sides are atomic, same symbol predicate and same number of terms, so we split into more equations.

$$equations = [\,(s = U), (s = G), (K = K)].$$

$$eq_1' = \big((s = U)\big) \circ \{\,\} = \big((s = U)\big)$$

We have a variable on one side, constant on the other:

$$sub = \{\,\} \circ (U = s) = \{U = s\}$$

next iteration;

$$eq_1' = (s = G) \circ \{ (U = s)\} = \ (s = G)$$

We have a variable on one side, constant on the other:

$$sub = \{ (U = s)\} \circ (G = s) = \{(U = s), (G = s)\}$$

next iteration;

$$eq_1' = (K = K) \circ \{(U = s), (G = s)\} = (K = K)$$

Same variable on both sides, we continue to the next iteration.

equations is empty, we return $\{(U = s), (G = s)\}$.

We can confirm:

$\big(\mathrm{t(s(s), G, s, p, t(K), s)} = \ \mathrm{t(s(G), G, s, p, t(K), U)}\big) \circ \{(U = s), (G = s)\} =$ $\mathrm{t(s(s), s, s, p, t(K), s)} = \ \mathrm{t(s(s), s, s, p, t}(K), \mathrm{s})$.

3.2 unify[p([v | [V | W]]), p([[v | V] | W])]

Init: $sub = \{\}, equations = \left[\left(\mathrm{p}([\mathrm{v}\,|[\mathrm{V}\,|\mathrm{W}]\right) = \ \mathrm{p}([[\mathrm{v} \mid \mathrm{V}] \mid \mathrm{W}]))\right]$

$$eq_1' = (\mathrm{p}([\mathrm{v}\,|[\mathrm{V}\,|\mathrm{W}]) = \ \mathrm{p}([[\mathrm{v} \mid \mathrm{V}] \mid \mathrm{W}])) \circ \{\}$$
$$= \ (\mathrm{p}([\mathrm{v}\,|[\mathrm{V}\,|\mathrm{W}] = \ \mathrm{p}([[\mathrm{v} \mid \mathrm{V}] \mid \mathrm{W}]))$$

Both sides are atomic, same symbol predicate and same number of terms, so we split into more equations.

Equations=[([v|[V|W]]=[[v|V]|W])]°{ } =[([v|[V|W]]=[[v|V]|W])]

Next iteration;

$$eq_1' = ([v\,|[V|W]] = [[v|V]|W])°\{\} = ([v\,|[V|W]] = [[v|V]|W])$$

We split the equations :

$$equations = [(v = [v|V]), ([V|W] = W)]$$

Next iteration;

$$eq_1' = (\,v = [v|V])°\{\} = (\,v = [v|V])$$

Now we can see that on the left side we have a constant, while in the right side we have a predicate with binary arity. Hence, the algorithm will terminate and return FAIL.