Assignment 4 - Theoretical questions

Question 1b

Let's define the equivalence of high-order function g and its CPS version g$ as follows: For any CPS-equivalent parameters f1…fn and f1$...fn$ **(g$ f1$...fn$ cont) is CPS-equivalent to (cont (g f1…fn))**. Now we are going to show that pipe$ is equivalent to pipe, by induction on the size of the list.

Base: N=1 (cont (pipe(f1$))) = (cont f1$) (pipe$ f1$ cont) = (cont (lambda (x cont2) (f1$ x cont2))) = (cont f1$)

Induction step: Assuming (pipe$ f1$ … fn$ cont) = (cont (pipe f1$ … fn$)).

(pipe$ (f1$ … fn$ fn+1$ cont)) =

= (pipe$ f2$ … fn+1$ (lambda (f2-n$) (cont (lambda (x cont2) (f1$ x (lambda (res) (fn2-n$ res cont2))))))) =

= (

(lambda (f2-n$) (cont (lambda (x cont2) (f1$ x (lambda (res) (fn2-n$ res cont2))))))

(pipe f2$ … fn+1$)

) =

= (cont (lambda (x cont2) ((pipe f2$ … fn+1$) x (lambda (res) (fn2-n$ res cont2))))) =

= (cont (f2-n$ (pipe f1$ f2$ … fn+1$)) =

= (cont (pipe f1$ … fn+1$)).

Question 2

b.

We will use reduce1-lzl when reducing the entire lazy list is needed and the lazy list is finite (gets to an empty list at some point). The function reduce1-lzl is not good for infinite lazy lists because it will never halt and can cause a stack overflow.

We will use reduce2-lzl when reduction until a known index (of a given infinite lazy list) is needed or when limiting the computation time is needed.

We will use the reduce3-lzl for tasks that require cumulative summaries or progressive calculations (of an infinite lazy list).

g.

Advantages:

- Performs less calculations when high precision is not required.
- Has the potential to use less memory at once.
- The precision of pi can be adjusted for the case and is not fixed in advance.

Disadvantages:

- Every time the value of pi is needed the calculations will be performed additional time.
- The value of pi might be inconsistent across the program.
- Generates a lot of closures. thus increases the usage of memory.

Question 3.1

1. unify[x(y(y), T, y, z, k(K), y), x(y(T), T, y, z, k(K), L)]

   A = x(y(y), T, y, z, k(K), y)

   B = x(y(T), T, y, z, k(K), L)

   S={T=y}

   AoS = x(y(y), y, y, z, k(K), y)

   BoS = x(y(y), y, y, z, k(K), L)

   S = {T=y, L=y}

   AoS = x(y(y), y, y, z, k(K), y)

   BoS = x(y(y), y, y, z, k(K), y)

   Answer: {T=y, L=y}

2. unify[f(a, M, f, F, Z, f, x(M)), f(a, x(Z), f, x(M), x(F), f, x(M))]
   A = f(a, M, f, F, Z, f, x(M))
   B = f(a, x(Z), f, x(M), x(F), f, x(M))
   S = {M = x(Z), Z=x(F), F=x(M)} = {M=x(x(F)), Z=x(x(M)), F=x(M)}
   = {M = x(x(x(M))), Z=x(x(M)), F=x(M)}
   Contradiction: M=x(x(x(M))).
3. unify[t(A, B, C, n(A, B, C),x, y), t(a, b, c, m(A, B, C), X, Y)]

   K = t(A, B, C, n(A, B, C),x, y)

L = t(a, b, c, m(A, B, C), X, Y)

S = {A=a, B=b, C=c, X=x, Y=y, n(A, B, C) = m(A, B, C)}

Contradiction = n(A, B, C) = m(A, B, C), functor n is different from functor m.

4. unify[z(a(A, x, Y), D, g), z(a(d, x, g), g, Y)]
   T = z(a(A, x, Y), D, g)
   K = z(a(d, x, g), g, Y)
   S = {D=g, A=d, Y=g}
   Answer: S = {D=g, A=d, Y=g}

Question 3.3

The tree is a success infinite tree because we can see it's infinite and it has at least one success path.

The diagram: