Assignment 4

Question 1

Section b

We will prove using induction on the number of the operands (n):

Base Case:

$n = 1$:

The function $pipe$ will get a_1 (the operand) and return it, so we will get $cont \circ a_1$.

The function $pipe\$$ will get a_1 and $cont$, so $(cdr\ fs)$ will be empty and $(car\ fs)$ will be a_1, therefore the function will return $cont \circ a_1$. In conclusion, they are equivalent.

Induction Hypothesis: We assume that for n operands the functions are equivilant.

Induction Step:

Assume that we give the functions $n + 1$ operands $(a_1\ a_2\ \ldots\ a_{n+1})$.

$pipe$ will return $cont \circ a_1 \circ a_2 \circ \cdots \circ a_{n+1}$.

$pipe\$$ will start by checking if $(cdr\ fs)$ is not empty (and indeed it is not), so it will return the value that comes back the call:

$$\left(pipe\$\ a_2\ a_3 \cdots a_{n+1}\ \left(lambda\ (pipe - res)\ \left(cont\ (compose\$\ a_1\ pipe - res)\right)\right)\right) =$$
$$\left(pipe\$\ a_2\ a_3 \cdots a_{n+1}\ \left(lambda\ (pipe - res)\ (cont \circ a_1\ pipe - res)\right)\right) \overset{IH}{=}$$
$$(cont \circ a_1 \circ\ a_2 \circ \cdots \circ a_{n+1})\ pipe - res$$

Question 2

Section d

We'll use reduce1-lzl when we are only interested in the final result and when we know the list is finite.

We'll use reduce2-lzl when the list may be infinite and/or when we are interested only in some prefix of it (for example when approximating some values up to a certain precision).

We'll use reduce3-lzl in the same scenario of reduce2-lzl, but this time we are also interested in the process (to draw a graph, maybe).

Section g

Advantages are that when using generate-pi-approximations we only calculate the desired values when we want them - no redundant calculations are made. We also do not use recursion - and thus saving a lot of space (on the call stack for example). Disadvantages may be that pi-sum gives us the ability to decide when to stop calculating depending on the actual fraction and not on the number of steps - thus it is more predictable and flexible.

Question 3

Section 1

1.

$$[x(y(y), T, y, z, k(K), y) = x(y(T), T, y, z, k(K), L], \{\}$$
$$[y(y) = y(T), y = L], \{\}$$
$$[y = T, y = L], \{\}$$
$$[y = L], \{y = T\}$$
$$[\,], \{\boldsymbol{y = T, y = L}\}$$

2.

$$\left[f\left(a, M, f, F, Z, f, x(M)\right) = f\left(a, x(Z), f, x(M), x(F), f, x(M)\right)\right], \{\}$$
$$[M = x(Z), F = x(M), Z = x(F)], \{\}$$
$$[F = x(M), Z = x(F)], \{M = x(Z)\}$$
$$[Z = x(F)], \{M = x(Z), F = x(M)\}$$
$$[\,], \left\{M = x\left(x(F)\right), F = x\left(x\left(x(F)\right)\right), Z = x(F)\right\}$$
$$- f \text{ appears in both sided of the equation} - \boldsymbol{failure}.$$

3.

$$[t(A, B, C, n(A, B, C), x, y) = t(a, b, c, m(A, B, C), X, Y)], \{\}$$
$$[A = a, B = b, C = c, n(A, B, C) = m(A, B, C), x = X, y = Y], \{\}$$
$$[n(A, B, C) = m(A, B, C), x = X, y = Y], \{A = a, B = b, C = c\}$$
$$- m \text{ and } n \text{ are not the same predicate} - \boldsymbol{failure}.$$

4.

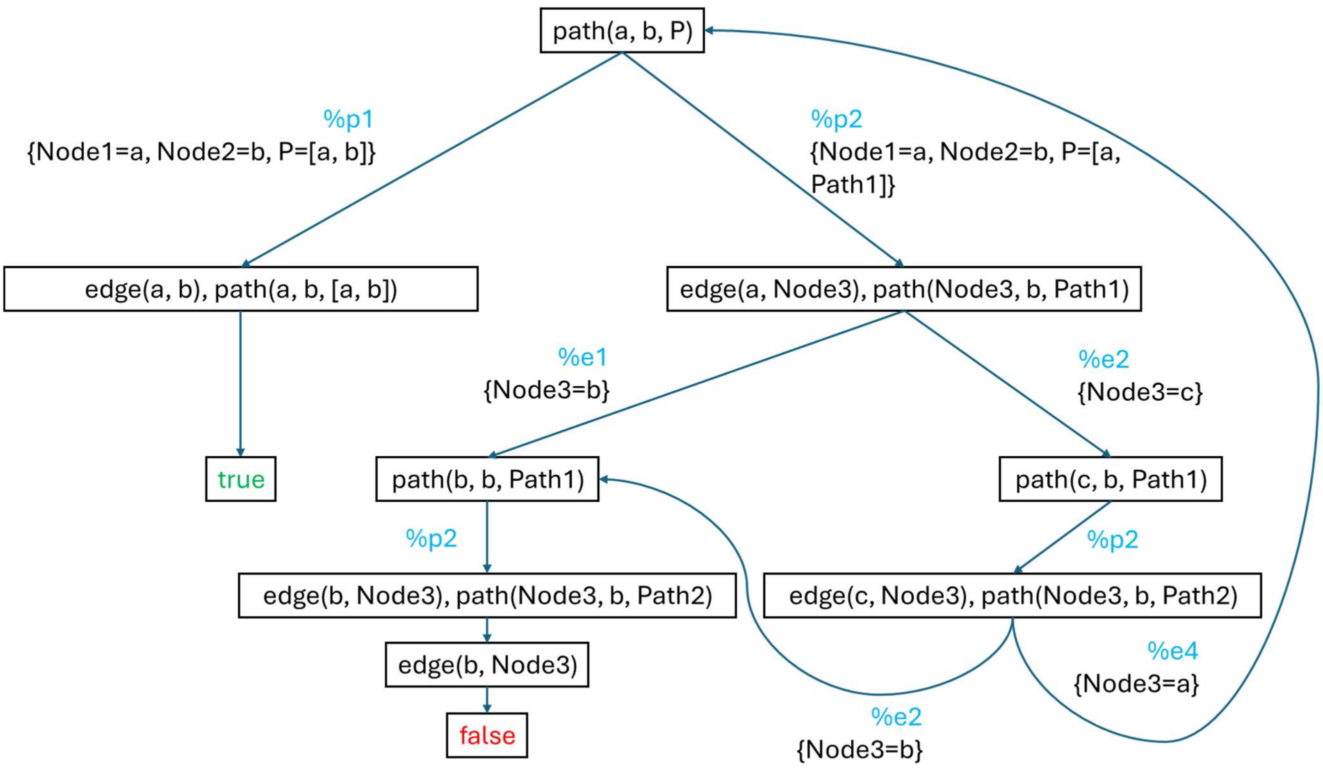
$$[z(a(A, x, Y), D, g) = z(a(d, x, g), g, Y)], \{\}$$
$$[a(A, x, Y) = a(d, x, g), D = g, g = Y], \{\}$$
$$[A = d, Y = g, D = g], \{\}$$
$$[\,], \{A = d, Y = g, D = g\}$$

Section 3

```
edge(a,b). %e1
edge(a,c). %e2
edge(c,b). %e3
edge(c,a). %e4

path(Node1, Node2, Path)
      :- edge(Node1, Node2), Path = [Node1, Node2]. %p1
path(Node1, Node2, Path)
      :- edge(Node1, Node3), path(Node3, Node2, Path1),
         Path = [Node1 | Path1].  %p2

path(a, b, P)
```



It is an infinite tree and also a success tree.