The purpose of this program was to write a variety of simple functions using SML. Some of these functions are integrated into others to demonstrate the building aspect of the language. The goal was to show how SML can accomplish sizable tasks with very little code through use of recursion. My code does exactly what was asked for in the description per file and nothing more or less. This is in the spirit of SML, a language that leaves little room for fluff or misinterpretation.

My difficulties on this assignment were mainly in trying to understand the recursive aspects. It was difficult enough to understand how a segment works and even more difficult to try and create one. Additionally I’m not used to debugging the language and so when I got something wrong it was difficult to figure out where my error was. This is very different from java, where I can hover over a syntax error and usually get a pretty good hint to what I messed up. An aspect that I really like was how the parts of the program build into each other.

fun choose 0 \_ = [nil]

=|choose \_ nil = nil

=|choose n (a::t) = (choose n t) @ map(fn x => a::x) (choose (n-1) t);

fun put a x nil = [x@[a]]

=|put a x (b::t) = (x@(a::b::t))::put a (x@[b]) t;

fun permute nil = [nil]

=|permute(a::t) = foldr(op@) nil (map(put a nil)(permute t));

fun powerset nil = [nil]

=| powerset (e::t) = (powerset t) @map (fn x => e::x) (powerset t);

fun choose 0 \_ = [nil]

=|choose \_ nil = nil

=|choose n (a::t) = (choose n t) @ map(fn x => a::x) (choose (n-1) t);

fun sfoldl \_ a nil = a

| sfoldl f a (x::t) = sfoldl f (f x a ) t;

fun cut a b = foldr(fn(c, d) => if c < a then c::d else d)[] b;

fun factorize n =

let val c = n

fun count(i, a) = if i > n then a else if Int.mod(n, i) = 0 then count(i+1, (i, n div i)::a) else count(i+1, a)

in count (2, nil);

fun product ( a::nil)(b::nil) = a \* b

=|product(a::A)(b::B)= a\*b + product A B;

fun listcon nil T = T

| listcon (a::s)nil = [a]:: listcon s nil

|listcon (a::s) (t::T) = (a ::t):: listcon s T;

fun transport nil = nil

|transport (r::nil) = listcon r nil

|transport ( r :: M) = listcon r ( transport M);

fun matrixMult (nil, A) = nil

| matrixMult (s::t, A) = (map (product s) (transport A)) :: matrixMult (t,A);