1. (Finite State Machine)

For part (1) the input alphabet is \{coin, pass\} and the output alphabet is \{green, red\}; the machine is shown in the left figure. For part (2), the input alphabet is \{coin, pass, reset\} and the output alphabet is \{green, red, alarm\}; the machine is shown in the right figure.

2. (Finite State Machine) Define the following predicates over any binary string \(x\):

\[
\begin{align*}
  p &= \text{every 0 is immediately followed by a 1 in } x \\
  q &= \text{the last item of } x \text{ is 0} \\
      &\quad \text{and every 0 in } x, \text{ except the very last, is immediately followed by a 1} \\
  r &= \text{there are two consecutive 0s in } x
\end{align*}
\]

I have attached the predicate names to the states in Fig 1.

![Figure 1: Accept binary strings in which a 0 is followed by a 1](image)

The theorems that have to be proved, one for each transition, are shown in Table 1. Additionally, \(p\) has to be proved for \(\epsilon\).

3. (Regular Expressions)

(a) \(0^* 1^* 2^*\) admits strings, such as 011, which are not strictly increasing.
Table 1: Verifications of state transitions

(b) zero = €|0
one = €|1
two = €|2

Observe that in an increasing string each symbol appears either once or none at all. Therefore, the desired expression is zero one two

4. (Types)

(a) charVal :: Int -> Char
(b) parallel :: ((Int, Int),(Int, Int)) -> ((Int, Int),(Int, Int)) -> Bool
(c) test :: (a -> Bool) -> a -> Bool
(d) tower :: Int -> b -> b -> b -> [(Int,b,b)]
(e) flatten :: [[a]] -> [a]

5. (Haskell Programming)

(a) The function takes a list of pairs as input and produces a pair of lists. Function transpose, given below, takes the first element of each input pair to form the first output list and the second elements to form the second output list.

transpose [] = ([],[])  
transpose ((a,b): xs) = ((a: ys), (b: zs))  
where (ys, zs) = try xs

(b) Define fib3 in terms of another function fibtriple.

fibtriple 0 = (0,1,2)
fibtriple n = (y, z, x+y+z)  
where (x,y,z) = fibtriple (n-1)
fib3 n = x  
where (x,y,z) = fibtriple n

(c) close (x:(y:[])) = x-y
close (x:(y:ys))
| x-y < close(y:ys) = x-y
| otherwise = close(y:ys)