1. (Finite State Machine Design)
   (a) See Figure 1.

   ![Figure 1: Binary string without 3 consecutive identical symbols](image)

   Figure 1: Binary string without 3 consecutive identical symbols

   (b) See Figure 2.

   ![Figure 2: accept if input contains “rare”](image)

   Figure 2: accept if input contains “rare”

2. (Reasoning about Finite State Machines)
   (a) See Figure 3.

   ![Figure 3: output every other symbol](image)

   Figure 3: output every other symbol

   (b) For arbitrary symbols \(a\) and \(b\), and string \(x\)

   \[ f(\epsilon) = \epsilon, \ f(a) = a, \ f(abx) = af(x) \]

   (c) Let \(\text{even}(x)\) denote that the length of \(x\) is even. Associate predicates \(p_0\) and \(p_1\) with states 0 and 1, where

   \[ p_0 \equiv \text{even}(x) \land y = f(x) \]
   \[ p_1 \equiv \neg\text{even}(x) \land y = f(x) \]
(d) \[ \text{evenl}(e) \land e = f(e) \]
\[ \neg\text{evenl}(x) \land y = f(x) \Rightarrow \text{evenl}(xa) \land y = f(xa), \text{for all } a \]
\[ \text{evenl}(x) \land y = f(x) \Rightarrow \neg\text{evenl}(xa) \land ya = f(xa), \text{for all } a \]

3. (Writing Recursive Programs)

(a) \[ \text{grade } [] = ([],[],[]) \]
\[ \text{grade}((\text{name},\text{score}): \text{xs}) \]
| score >= 90 = ((\text{name:a}),b, c) |
| score >= 80 = (a,(\text{name:b}), c) |
| otherwise = (a,b, (\text{name:c})) |
where (a,b,c) = \text{grade} \text{xs}

(b) \[ \text{suffix } [] = [[]] \]
\[ \text{suffix} (x:xs) = (x:xs):(\text{suffix} \text{xs}) \]

(c) \[ \text{cart1} x [] = [] \]
\[ \text{cart1} x (y:ys) = (x,y) : (\text{cart1} x \text{ys}) \]
\[ \text{cart [] ys} = [] \]
\[ \text{cart} (x:xs) \text{ ys} = (\text{cart1} x \text{ys}) ++ (\text{cart} \text{xs} \text{ys}) \]

(d) We define function scan that has three arguments: (1) the part of the string that has already been scanned, call it left, (2) the left paren count − the right paren count over left, call it n, and (3) the part of the string that remains to be scanned, call it right. Function scan returns True iff left ++ right is balanced.

Then, balanced xs = scan [] 0 xs.

In defining scan we will ensure that n is non-negative. The function is easy to write:

\[ \text{scan left n ""} = n == 0 \]
\[ \text{scan left 0 ('}':xs) = \text{False} \]
\[ \text{scan left n ('}':xs) = \text{scan} (\text{left ++ ""}) (n-1) \text{ xs} \]
\[ \text{scan left n ('}':xs) = \text{scan} (\text{left ++ ""}) (n+1) \text{ xs} \]

Now observe that left is used only in computing its own next value; it does not affect the other two arguments in the last two clauses, nor the result in the first two clauses. So, we can eliminate left altogether.

\[ \text{scan n ""} = n == 0 \]
\[ \text{scan 0 ('}':xs) = \text{False} \]
\[ \text{scan n ('}':xs) = \text{scan} (n-1) \text{ xs} \]
\[ \text{scan n ('}':xs) = \text{scan} (n+1) \text{ xs} \]

Then,

\[ \text{balanced xs} = \text{scan} 0 \text{ xs} \]
4. (Properties of Recursive programs) The proof of $rr(\text{lr } xs) = xs$ is by case discrimination, $xs = []$ and $xs \neq []$. Note that we never employ an inductive hypothesis; all induction are buried in the given facts (0–3).

- $xs = []$:
  We have to show: $rr(\text{lr } []) = []$

  \[
  rr(\text{lr } []) \\
  = \{ \text{lr } [] = [] \} \\
  rr [] \\
  = \{ rr [] = [] \} \\
  []
  \]

- Input list is non-empty:
  We have to show $rr(\text{lr } (x:xs)) = (x:xs)$

  \[
  rr(\text{lr } (x:xs)) \\
  = \{ \text{from definition of lr, } \text{lr } (x:xs) = xs ++ [x] \} \\
  rr(xs ++ [x]) \\
  = \{ \text{definition of } rr \} \\
  y:(\text{rev } ys) \text{ where } y:ys = \text{rev } (xs ++ [x]) \\
  = \{ \text{from given fact (2): } \text{rev } (xs ++ [x]) = (\text{rev } [x]) ++ (\text{rev } xs) \} \\
  y:(\text{rev } ys) \text{ where } y:ys = (\text{rev } [x]) ++ (\text{rev } xs) \\
  = \{ \text{from given fact (0): } \text{rev } [x] = [x] \} \\
  y:(\text{rev } ys) \text{ where } y:ys = [x] ++ (\text{rev } xs) \\
  = \{ \text{from given fact (3): } [x] ++ (\text{rev } xs) = x:(\text{rev } xs) \} \\
  y:(\text{rev } ys) \text{ where } y:ys = x:(\text{rev } xs) \\
  = \{ \text{substituting for } y \text{ and } ys \} \\
  x:(\text{rev } (\text{rev } xs)) \\
  = \{ \text{from given fact (1): } \text{rev } (\text{rev } xs) = xs \} \\
  x:xs
  \]