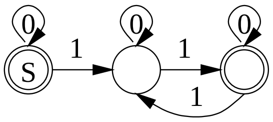
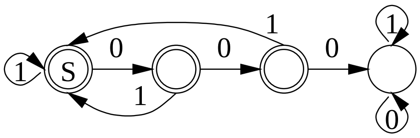
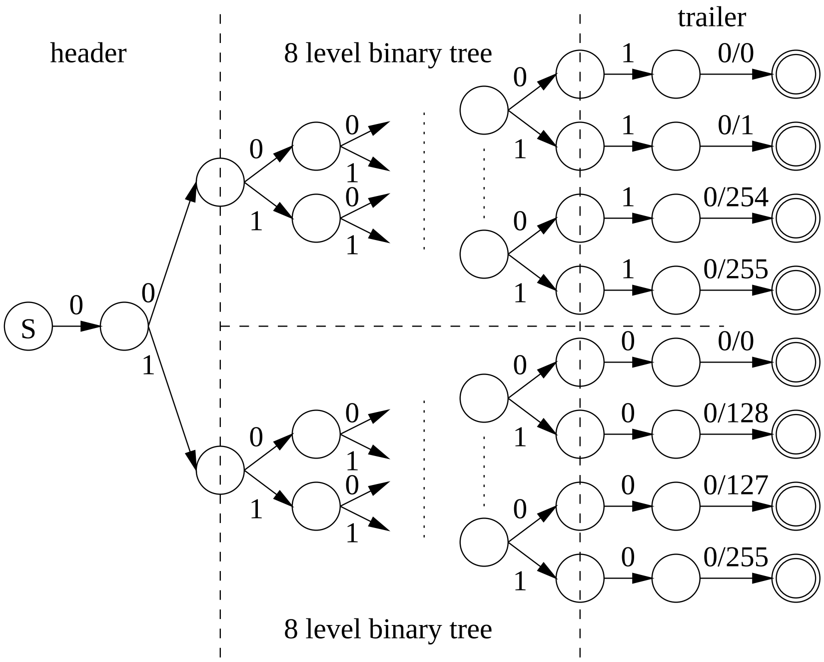
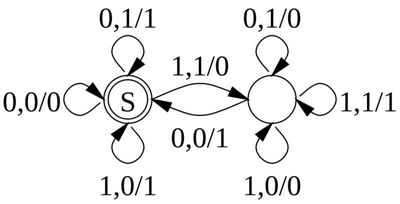
COMP 280 : Assignment 11 - Sample Solutions

1. 

2. 

3. 

This FA outputs the barcode data as a decimal number. To just recognize valid barcodes, the two binary trees could be replaced with a simple straight line eight transition long FA that would accept any sequence of eight binary digits, and all the trailers could then be collapsed into one.

4. 

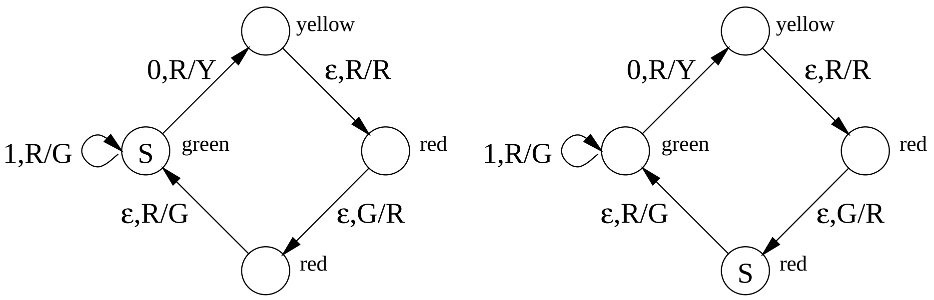
We assume that no overflow is generated. If an overflow might be generated, we assume that the inputs are preceded by zeros (which is why the "has carry" state is not a final state).

5. Proof by induction on the length of y.

   Let $x = \langle i_1, \ldots, i_k \rangle$.

   Base case: $y = \langle j \rangle$. Then $R^*(q, xy) = R^*(q, \langle i_1, \ldots, i_k, j \rangle) = R(R^*(q, \langle i_1, \ldots, i_k \rangle), j)$ (by definition) $= R^*(R^*(q, \langle i_1, \ldots, i_k \rangle), \langle j \rangle)$ (by the base case of the definition) $= R^*(R^*(q, x), y)$.

   Induction: let $y = \langle j_1, \ldots, j_n \rangle$, and assume $R^*(q, xy) = R^*(R^*(q, x), y)$. Then $R^*(q, xy\langle j_{n+1} \rangle)$ $= R(R^*(q, xy), j_{n+1})$ (by definition) $= R(R^*(R^*(q, x), y), j_{n+1})$ (by the induction hypothesis) $=$ $R^*(R^*(q, x), y\langle j_{n+1} \rangle)$ (by definition).

6. (a) 

The two automata are the same, except for the starting state. 0 stands for “no car waiting”, 1 for “car waiting”, G for green, Y for yellow, R for red. The automata assume that there are no infinite flow of cars (i.e. there has to be no car waiting from time to time), and that the input corresponding to the color of each light can be generated as many times as necessary (i.e. input is generated as needed). There are two “red” states, to ensure that the one light becomes green before the other light becomes green again (i.e. prevent one light from continuously changing color while the other one is stuck at red). Note that, from the way the automata are designed, the lights will continuously change color when there are no car present. Note also that drivers are well behaved: they stop at yellow lights...

(b) Condition: the traffic light is safe if, starting from the start state, there is no input sequences I_1 and I_2 such that the run of the intersection of the two automata on I_1 and I_2 contains a state from which both automata output the color green.

Proof sketch (not required): a light can only go from one state to a different state either if the other light is red, or if it is itself in the first red state and the other light is green, in which case it remains red (by going to the second red state). So if the current state is safe, only transitions to safe states can be made. Since the initial state is safe, all reachable states are then safe.

(c) Note: instead of having one transition table for both automata, one could write two separate transition tables and use a helper rule to use the two tables simultaneously.

```
% transition(first-light-state,
%               car-present-at-first-light,
%                  second-light-state,
%                     car-present-at-second-light,
%                        the same four things but after the transition...
%
% Note that each automata from question (6a) does not know
% the state of the other automata, only the color of the light.
% Since states r1 and r2 both correspond to the color red, we
% have to duplicate each rule so that the state (r1 or r2) of
% the other automaton does not matter.
% We then have to duplicate again, because the transition of
% one automaton is independant of whether there is a car waiting
% at the other red light...
% Hence the long transition table...
%
% first FA
% g -> y
transition(g, [0], r1, [0], y, [], r1, [0]).
transition(g, [0], r1, [1], y, [], r1, [1]).
transition(g, [0], r2, [0], y, [], r2, [0]).
transition(g, [0], r2, [1], y, [], r2, [1]).

% g -> g (car disappears)
transition(g, [1], r1, [0], g, [], r1, [0]).
```

```
transition(g, [1], r1, [1], g, [], r1, [1]).
transition(g, [1], r2, [0], g, [], r2, [0]).
transition(g, [1], r2, [1], g, [], r2, [1]).

% y -> r1
transition(y, [0], r1, [0], r1, [0], r1, [0]).
transition(y, [0], r1, [1], r1, [0], r1, [1]).
transition(y, [0], r2, [0], r1, [0], r2, [0]).
transition(y, [0], r2, [1], r1, [0], r2, [1]).

% y -> r1 (car present)
transition(y, [1], r1, [0], r1, [1], r1, [0]).
transition(y, [1], r1, [1], r1, [1], r1, [1]).
transition(y, [1], r2, [0], r1, [1], r2, [0]).
transition(y, [1], r2, [1], r1, [1], r2, [1]).

% r1 -> r2
transition(r1, [0], g, [0], r2, [0], g, [0]).
transition(r1, [0], g, [1], r2, [0], g, [1]).

% r1 -> r2 (car present)
transition(r1, [1], g, [0], r2, [1], g, [0]).
transition(r1, [1], g, [1], r2, [1], g, [1]).

% r2 -> g
transition(r2, [0], r1, [0], g, [0], r1, [0]).
transition(r2, [0], r1, [1], g, [0], r1, [1]).
transition(r2, [0], r2, [0], g, [0], r2, [0]).
transition(r2, [0], r2, [1], g, [0], r2, [1]).

% r2 -> g (car present)
transition(r2, [1], r1, [0], g, [1], r1, [0]).
transition(r2, [1], r1, [1], g, [1], r1, [1]).
transition(r2, [1], r2, [0], g, [1], r2, [0]).
transition(r2, [1], r2, [1], g, [1], r2, [1]).

% second FA
% g -> y
transition(r1, [0], g, [0], r1, [0], y, []).
transition(r1, [1], g, [0], r1, [1], y, []).
transition(r2, [0], g, [0], r2, [0], y, []).
transition(r2, [1], g, [0], r2, [1], y, []).

% g -> g (car disappears)
transition(r1, [0], g, [1], r1, [0], g, []).
transition(r1, [1], g, [1], r1, [1], g, []).
transition(r2, [0], g, [1], r2, [0], g, []).
transition(r2, [1], g, [1], r2, [1], g, []).

% y -> r1
transition(r1, [0], y, [0], r1, [0], r1, [0]).
transition(r1, [1], y, [0], r1, [1], r1, [0]).
transition(r2, [0], y, [0], r2, [0], r1, [0]).
```

```
transition(r2, [1], y, [0], r2, [1], r1, [0]).

% y -> r1 (car present)
transition(r1, [0], y, [1], r1, [0], r1, [1]).
transition(r1, [1], y, [1], r1, [1], r1, [1]).
transition(r2, [0], y, [1], r2, [0], r1, [1]).
transition(r2, [1], y, [1], r2, [1], r1, [1]).

% r1 -> r2
transition(g, [0], r1, [0], g, [0], r2, [0]).
transition(g, [1], r1, [0], g, [1], r2, [0]).

% r1 -> r2 (car present)
transition(g, [0], r1, [1], g, [0], r2, [1]).
transition(g, [1], r1, [1], g, [1], r2, [1]).

% r2 -> g
transition(r1, [0], r2, [0], r1, [0], g, [0]).
transition(r1, [1], r2, [0], r1, [1], g, [0]).
transition(r2, [0], r2, [0], r2, [0], g, [0]).
transition(r2, [1], r2, [0], r2, [1], g, [0]).

% r2 -> g (car present)
transition(r1, [0], r2, [1], r1, [0], g, [1]).
transition(r1, [1], r2, [1], r1, [1], g, [1]).
transition(r2, [0], r2, [1], r2, [0], g, [1]).
transition(r2, [1], r2, [1], r2, [1], g, [1]).

%%

append([],L,L).
append([X|L1],L2,[X|L12]) :- append(L1,L2,L12).

% lights(NS-traffic, NS-state, EW-traffic, EW-state)

% note: the two FA don't have final states, because
% lights are supposed to run forever. Here we only
% have finite lists, so we assume that all states are
% final states, and we stop when a list becomes empty
lights(_, _, [], _).
lights([], _, _, _).

%
lights([H1|T1], S1, [H2|T2], S2) :-
    transition(S1, [H1], S2, [H2], New_S1, New_H1_l, New_S2, New_H2_l),
    append(New_H1_l, T1, New_T1),
    append(New_H2_l, T2, New_T2),
    lights(New_T1, New_S1, New_T2, New_S2).

% test cases

% lights([0,0,0,1,0,0,0], g, [0,0,0,0,0,0,0], r2).
% lights([1,0,0,1,1,1,0,1,0], g, [0,0,0,1,1,1,1,1,0,1,0,1], r2).
```

```
%
% start from an intermediate state, but state is safe
% lights([0,0,0,1,0,0,0], r2, [0,0,0,0,0,0,0], r2).
%
% start from an intermediate state, and get stuck waiting for a green light
% lights([0,0,0,1,0,0,0], r1, [0,1,0,1,1,0,0,0,1,1,0], r1).
```