7 (100 PTS.) Fool me once.

You are given a DFA $M = (Q, \Sigma, \delta, s, A)$ over the alphabet $\Sigma = \{0, 1\}$. The purpose of this question is to describe an efficient (i.e., polynomial time) algorithm to compute the minimal automata (in the number of stats) M' such that $L(M') = L(M)$.

7.A. (20 PTS.) Two states $q, q' \in Q$ are ***equivalent*** if

$$\forall w \in \Sigma^* \qquad \delta(q, w) \in A \iff \delta(q', w) \in A.$$

Describe how to check (efficiently [i.e., polynomial running time in the input size]) if L_q and $L_{q'}$ are the equal, where

$$L_q = \{w \in \Sigma^* \mid \delta(q, w) \in A\}$$

is the ***language of*** q. Using this, describe how to decide if q and q' are equivalent. (Hint: How do you decide if two automatas accept the same language?)

Two languages, A, B are equivalent if L(A) ⊆ L(B) and L(B) ⊆ L(A).
To show L(A) ⊆ L(B), we need to show L(A) ∩ $L_{Complement}$(B) is empty.
To show L(B) ⊆ L(A), we need to show L(B) ∩ $L_{Complement}$(A) is empty.
If both L(A) ∩ $L_{Complement}$(B) and L(B) ∩ $L_{Complement}$(A) are empty, then L(A) and L(B) are equivalent.

To check if q and q' are equivalent, we would need to check if L(q) ∩ $L_{Complement}$(q') and L(q') ∩ $L_{Complement}$(q) are both empty.

7.B. (50 PTS.) Given that q and q' are equivalent, in the given DFA $M = (Q, \Sigma, \delta, s, A)$, and $q \neq q'$, describe formally how to construct (efficiently) a new DFA M' with $n - 1$ states, such that $L(M') = L(M)$, where $n = |Q|$. ***Prove*** that your construction is correct.
(Hint: For the proof of correctness, you need to prove that the languages of the states of the new automata are the same as they were in the original automata. This probably should be done by induction on the length of the strings.)

Call the set of states that jump to q, PQ, and the set of states that jump to q', PQ'.

Let the states in PQ jump to q' instead of q.

Since q and q' are equivalent, that is, ∀w ∈ ∑*, δ(q, w) ∈ A <==> δ(q', w) ∈ A, we know that any state going to q will not have its language changed by going to q'.

This also means that no state in the DFA can jump to q, allowing us to remove it from the DFA without changing the language of the new DFA.

At no point is the language of the new DFA altered from the original DFA. Therefore, the new construction is correct.

7.C. (30 PTS.) Using the above, and only the above, describe how to construct a DFA M' with $L(M') = L(M)$, such that no two states in M' are equivalent. How many invocations of the algorithm of part B your algorithm requires in computing M'?
(The Myhill-Nerode theorem implies M' is the minimal automata for $L(M)$.)

A DFA has a finite amount of states. Let n equal the total number of states.

Let each state be ordered in a list with an arbitrary permutation.

Step 1: Starting from the first state, check if the current state is equivalent to any of the following states. If it is, remove the following state from the set using Part B's algorithm.

Step 2: After checking the current state against all of the following states, advance the current state to the next state in the list. Repeat Step 1 until the current state is the last state in the list.

Give a list of n size, this will occur at most $n^2 / 2$ times, or $O(n^2)$.

8 (100 PTS.) Fool me twice.

Let $\Sigma = \{0, 1, 2\}$. Consider the language L of all strings $w \in \Sigma^*$ where $\#_0(w) = \#_1(w) + \#_2(w)$.

8.A. (30 PTS.) Prove that this language is CFG by providing a grammar G for it.

V: { A }
Σ: { 0, 1, 2 }
R: { A —> e, A —> 0A1, A —> 1A0, A —> 0A2, A —> 2A0, A —> AA }
S: { A }

8.B. (70 PTS.) Prove that your grammar indeed yields the desired language (i.e., prove that $L(G) \subseteq L$ and $L \subseteq L(G)$).

https://youtu.be/dOP1ASa6jZg

Proof By Induction:
Base Case:
If a single substitution in L(G) is applied, it can only result in e.
e is also on L, as $\#_0(w) = 0 = \#_1(w) + \#_2(w) = 0 + 0$
The base case holds.
Inductive Hypothesis:
Let n = k, where k >= 1. Assume that for all strings w with at most n substitutions of L(G), w ∈ L.
We want to show that for any string w with k + 1 substitutions, w ∈ L.
Inductive Step:
Since n + 1 > 1, each derivation consists of at least 2 production rules, and the only production rules whose bodies have variables are either A —> 0A1, A —> 1A0, A —> 0A2, or A —> 2A0.
By the Inductive Hypothesis, we know that any derivation with n substitutions is contained in the language L.
That is, any string in L with n derivations can be expanded into a tree of production rules in L(G).
The production rule A —> AA lets us add (0 and 1) or (0 and 2) to any two indices of the tree, and in effect, any two indices of the initial string.
This means that for any string w with k + 1 substitutions, w ∈ L.
Therefore, for all strings w, w ∈ L if and only if w ∈ L(G).

Btw, I would appreciate it if you could tell me how I would prove it using L(G) ⊆ L and L ⊆ L(G), because I couldn't find a way to prove that $L \cap L_{Complement}(G)$ is empty.