Exercise 3.2.4: Convert the following regular expressions to NFA’s with \( \epsilon \)-transitions.

* a) \( 01^* \).

* b) \( (0 + 1)01 \).

* c) \( 00(0 + 1)^* \).

Exercise 3.2.5: Eliminate \( \epsilon \)-transitions from your \( \epsilon \)-NFA’s of Exercise 3.2.4. A solution to part (a) appears in the book’s Web pages.

Exercise 3.2.6: Let \( A = (Q, \Sigma, \delta, q_0, \{q_f\}) \) be an \( \epsilon \)-NFA such that there are no transitions into \( q_0 \) and no transitions out of \( q_f \). Describe the language accepted by each of the following modifications of \( A \), in terms of \( L = L(A) \):

* a) The automaton constructed from \( A \) by adding an \( \epsilon \)-transition from \( q_f \) to \( q_0 \).

* b) The automaton constructed from \( A \) by adding an \( \epsilon \)-transition from \( q_0 \) to every state reachable from \( q_0 \) (along a path whose labels may include symbols of \( \Sigma \) as well as \( \epsilon \)).

* c) The automaton constructed from \( A \) by adding an \( \epsilon \)-transition to \( q_f \) from every state that can reach \( q_f \) along some path.

* d) The automaton constructed from \( A \) by doing both (b) and (c).

Exercise 3.2.7: There are some simplifications to the constructions of Theorem 3.7, where we converted a regular expression to an \( \epsilon \)-NFA. Here are three:

1. For the union operator, instead of creating new start and accepting states, merge the two start states into one state with all the transitions of both start states. Likewise, merge the two accepting states, having all transitions to either go to the merged state instead.

2. For the concatenation operator, merge the accepting state of the first automaton with the start state of the second.

3. For the closure operator, simply add \( \epsilon \)-transitions from the accepting state to the start state and vice-versa.

Each of these simplifications, by themselves, still yield a correct construction; that is, the resulting \( \epsilon \)-NFA for any regular expression accepts the language of the expression. Which subsets of changes (1), (2), and (3) may be made to the construction together, while still yielding a correct automaton for every regular expression?

Exercise 3.2.8: Give an algorithm that takes a DFA \( A \) and computes the number of strings of length \( n \) (for some given \( n \), not related to the number of states of \( A \)) accepted by \( A \). Your algorithm should be polynomial in both \( n \) and the number of states of \( A \). Hint: Use the technique suggested by the construction of Theorem 3.4.