Please note that there is more than one way to answer most of these questions. The following only represents a sample solution.

(1) (a) Construct a state diagram for a DFA $M_1$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_1 = \{w : \text{the number of } a\text{s in } w \text{ is even}\}$.

Answer:

(b) Construct a state diagram for a DFA $M_2$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_2 = \{w : \text{the number of } b\text{s in } w \text{ is at most } 3\}$.

Answer:

(c) Using the method in Theorem 1.25 and your answers to the (a) and (b) parts, construct a state diagram for a DFA $M_3$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_3 = \{w : \text{the number of } a\text{s in } w \text{ is even or the number of } b\text{s in } w \text{ is at most } 3\}$.

Answer:
(d) Using the method in the footnote on page 46 and your answers to the (a) and (b) parts, construct a state diagram for a DFA $M_4$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_4 = \{w : \text{the number of } a$s in } w \text{ is even and the number of } b$s in } w \text{ is at most } 3\}.

Answer:

(e) By whatever method you like, construct a state diagram for a DFA $M_5$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_5 = \{wz : \text{the number of } a$s in } w \text{ is even and the number of } b$s in } z \text{ is at most } 3\}.

Answer:

Note that the language $L_5$ is equivalent to $\{w : \text{the number of } a$s in } w \text{ is even, or the number of } a$s in } w \text{ is odd and there is at most } 3 \text{ } b$s in } w \text{ after the last } a\}.$
Consider the three languages $L_1 \cup L_2$, $L_1 \cap L_2$, and $L_1 L_2$. For each pair of these, is one of the languages contained in the other? Is either properly contained in the other? Explain.

Solution:

$L_1 \cap L_2$ is properly contained (and hence contained) in $L_1 \cup L_2$. To see containment note that if $w \in L_1 \cap L_2$ then $w \in L_1$ so $w \in L_1 \cup L_2$. To see the containment is proper note the string $a$ is in $L_2$ and is thus in $L_1 \cup L_2$, but $a$ does not have any even number of $a$s and is thus not in $L_1$ and so it cannot be in $L_1 \cap L_2$. Since $L_1 \cap L_2$ is properly contained in $L_1 \cup L_2$, $L_1 \cup L_2$ is not contained in $L_1 \cap L_2$.

$L_1 \cup L_2$ is properly contained in $L_1 L_2$. To see containment note that if $w \in L_1 \cup L_2$ then $w \in L_1$ or $w \in L_2$. If $w \in L_1$, then $we = w \in L_1 L_2$ since $e \in L_2$. Similarly, if $w \in L_2$, then $ew = w \in L_1 L_2$ since $e \in L_1$. To see the containment is proper, note that $bbbab \in L_1 L_2$, since $bbb \in L_1$ (has 0 $a$s) and $ab \in L_2$ (has 1 $b$), but $bbbab$ does not have an even number of $a$s and is thus not in $L_1$ and $bbbab$ has more than three $b$s so it is also not in $L_2$. Thus $bbbab$ cannot be in $L_1 \cup L_2$. Since $L_1 \cup L_2$ is properly contained in $L_1 L_2$, $L_1 L_2$ is not contained in $L_1 \cup L_2$.

$L_1 \cap L_2$ is properly contained in $L_1 L_2$. One way to see this is to realize that proper containment is a transitive property, which means since $L_1 \cap L_2$ is properly contained in $L_1 \cup L_2$ and $L_1 \cup L_2$ is properly contained in $L_1 L_2$, then $L_1 \cap L_2$ must be properly contained in $L_1 L_2$.

Another way is to proceed as we did above. To see containment note that if $w \in L_1 \cap L_2$ then $w \in L_1$ and $we = w \in L_1 L_2$ since $e \in L_2$. To see the containment is proper, note that $ea = a \in L_1 L_2$, but $a$ does not have an even number of $a$s and is thus not in $L_1$ and so it cannot be in $L_1 \cap L_2$. Since $L_1 \cap L_2$ is properly contained in $L_1 L_2$, $L_1 L_2$ is not contained in $L_1 \cap L_2$. 

3
(2) (a) Construct a state diagram for an NFA $M_1$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_1 = \{w :$ the number of $a$s in $w$ is a multiple of 3, and $w$ starts and ends with the same letter $\}$. Use as few states as you can! (Note: we assume that $w$ has at least one letter, in order to be able to refer to the ‘first’ letter and the ‘last’ letter.)

Answer:

![State Diagram for NFA $M_1$]

(b) Using the method of Theorem 1.39 and your answer to the (a) part, construct a DFA for $L_1$.

Answer:

Only the states reachable from the start state are included in the state diagram.

![State Diagram for DFA]

4
(c) Construct a state diagram for an NFA $M_2$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_2 = \{w : w$ starts with an $a$ and ends with a $b\}$. Use as few states as you can!

Answer:

![Diagram](image1)

(d) Using the method of Theorem 1.39 and your answer to the (c) part, construct a DFA for $L_2$.

Answer:

Only the states reachable from the start state are included in the state diagram.

![Diagram](image2)
(e) Using the method in Theorem 1.45 and your answers to the (a) and (c) parts, construct a state diagram for an NFA $M_3$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_3 = L_1 \cup L_2$.

Answer:

\begin{figure}
\centering
\includegraphics[width=\textwidth]{diagram1}
\end{figure}

(f) Using the method of Theorem 1.47 and your answers to the (a) and (c) parts, construct a state diagram for an NFA $M_4$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_4 = L_1L_2$.

Answer:

\begin{figure}
\centering
\includegraphics[width=\textwidth]{diagram2}
\end{figure}
(g) By whatever method you like, construct a state diagram for an NFA $M_5$ to recognize the language over $\Sigma = \{a, b\}$ defined by $L_5 = L_1 \cap L_2$. Use as few states as you can! Explain your method.

**Answer:**

Since for a string $w$ to be in $L_1$ it must begin and end with the same letter, while to be in $L_2$ it must begin with an $a$ and end with a $b$, there are no strings that can be in both $L_1$ and $L_2$. Therefore, $L_5 = L_1 \cap L_2 = \emptyset$, whose state diagram is given below.

![State Diagram](image)

(3) Sipser 1.15, page 85.

**Answer:**

The main difference between the construction proposed in the problem and the one given in Theorem 1.49 is that the construction in this problem does not create a new start state. That is what we will use in the construction of a counterexample. Let $N_1$ be the machine given by the following state diagram:

![State Diagram](image)

Then $A_1 = \{a^ib : i \geq 0\}$, in other words, 0 or more as followed by a single $b$. Note that since everything in $A_1$ ends with a $b$, then everything in $A_1^*$ except for $\epsilon$ also ends in a $b$. In particular, the string $a \notin A_1^*$. Following the construction given in the problem, we generate the NFA $N$ given by its state diagram:

![State Diagram](image)

It is clear from the state diagram that $a$ is in the language recognized by $N$, $L(N)$. Since $a \in L(N)$ but $a \notin A_1^*$, $L(N) \neq A_1^*$. Therefore $N$ does not recognize $A_1^*$. 
(4) Sipser 1.16, page 86.

Answer:

(a)

(b) Only the states reachable from the start state are included in the state diagram.

(5) Sipser 1.24 and 1.25, page 87.

Answer:

1.24

a. 000
b. 111
c. 011
d. 0101
e. 1
f. 1111
g. 110110
h. \(\epsilon\)
A finite state transducer (FST) is a 5-tuple \((Q, \Sigma, \delta, q_0, \Gamma)\), where

1. \(Q\) is a finite set called the **states**,  
2. \(\Sigma\) is a finite set called the **input alphabet**,  
3. \(\delta : Q \times \Sigma \rightarrow Q \times \Gamma\) is the **transition function**,  
4. \(q_0 \in Q\) is the **start state**, and  
5. \(\Gamma\) is a finite set called the **output alphabet**.

A **computation** of an FST \(F = (Q, \Sigma, \delta, q_0, \Gamma)\) on input \(w = w_1w_2 \ldots w_n \in \Sigma^*\) outputs \(x = x_1x_2 \ldots x_n \in \Gamma^*\) iff there exists a sequence of states \(r_0, r_1, \ldots, r_n\) in \(Q\) such that

1. \(r_0 = q_0\), and  
2. \(\delta(r_i, w_{i+1}) = (r_{i+1}, x_{i+1})\) for \(0 \leq i < n\).

Suggestion: Another exercise that is well worth doing is Sipser 1.14, page 85. This is not to be handed in for grading, however.

**Solution:**

**a.**

Let \(M = (Q, \Sigma, \delta, q_0, F)\) be a DFA such that \(L(M) = B\). Also, let \(M' = (Q, \Sigma, \delta, q_0, Q \setminus F)\). This problem asks us to show that \(\overline{B} = L(M')\). \(w_1w_2 \ldots w_n = w \in \overline{B}\) iff there is a computation consisting of states \(r_0, r_1, \ldots, r_n\) in \(Q\) such that \(r_0 = q_0\), \(\delta(r_i, w_{i+1}) = r_{i+1}\) for \(0 \leq i < n\), but such that \(r_n \notin F\) iff \(r_n \in Q \setminus F\) iff \(r_0, r_1, \ldots, r_n\) is an accepting computation for \(M'\) iff \(w \in L(M')\).

This shows that \(\overline{B} = L(M')\). Since for every regular language there exists a finite automaton that recognizes its complement (constructed by \(M'\)), we conclude that the class of regular languages is closed under complement.
b.

Let $M$ be the NFA given by the following state diagram:

![State Diagram](image)

Then $C = L(M) = \{w : w \text{ ends with a } b\}$. Therefore $\overline{C} = \{w : w \text{ does not end with a } b\}$. In particular the string $b \notin \overline{C}$. Swapping the accept and non-accept states of $M$ gives the NFA $M'$ given by the following state diagram:

![State Diagram](image)

Clearly, $L(M') = \Sigma^*$, that is every string over $\Sigma$ which includes $b$. However, then $b \in L(M')$, but $b \notin \overline{C}$. Therefore, $L(M') \neq \overline{C}$. Thus, swapping the accepting and non-accepting states of an NFA does not give an NFA that recognizes the complement of the original.

However, the class of languages recognized by NFAs is closed under complement. The class of languages accepted by NFAs is the same as the class of languages accepted by DFAs, the class of regular languages as shown by Theorem 1.39. Part (a) of this problem showed that regular languages are closed under complement. Thus, the class of languages recognized by NFAs is closed under complement. (One could always covert the NFA to an equivalent DFA and then swap the accepting and non-accepting states to produce a DFA (and hence NFA) that recognizes the complement of the original NFA).