2.1.2. Construct DFA over $\Sigma = \{0, 1\}$ that accepts all strings in which 110 does \textit{not} appear as a substring.

The following DFA accepts strings that contain a 110 substring.

We can swap accepting and rejecting states to obtain the following DFA, which accepts all strings that do not contain a 110 substring.

2.1.3. Construct a DFA over $\Sigma = \{0, 1\}$ that accepts all strings that contain at least 5 ones.

2.1.7. Construct a DFA over $\Sigma = \{0, 1\}$ that accepts strings that begin with 1 and end with 0.
2.2.1. Construct an NFA over \( \Sigma = \{0, 1\} \) that has three states and accepts strings that end with 10.

2.2.3. Construct an NFA over \( \Sigma = \{0, 1\} \) that has six states and accepts strings that contain an odd number of ones or exactly two zeros.

2.3.1. Construct an NFA over \( \Sigma = \{0, 1\} \) that accepts string that contain the substring 11001.
2.3.2. Construct an NFA over $\Sigma = \{0, 1\}$ that accepts strings that have length at least two and do not end in 10.

Here are two DFAs, which are also NFAs, the second managing a state reduction by one:

And here is a true NFA:

2.3.3 Construct an NFA over $\Sigma = \{0, 1\}$ that accepts strings that begin with a one or end with a zero.
2.4. Convert the following NFA to an equivalent DFA:

```
\[
\begin{array}{c|cc}
\delta & a & b \\
\hline
\Rightarrow * & \{1\} & \{1, 2\} \\
& \{1, 2\} & \{1, 2\} \\
& \{2\} & \phi \\
& \phi & \{1\}
\end{array}
\]
```

2.5. Convert the following NFA to an equivalent DFA:

```
\[
\begin{array}{c|ccc}
\delta & a & b \\
\hline
\Rightarrow * & \{1, 2\} & \{1, 2, 3\} \\
& \{1, 2, 3\} & \{2, 3\} \\
& \{2, 3\} & \{3\} \\
& \{3\} & \phi \\
& \phi & \{3\}
\end{array}
\]
```

2.6. Convert the following NFA to an equivalent DFA:

```
\[
\begin{array}{c|cc}
\delta & a & b \\
\hline
\Rightarrow * & \{0, 1\} & \{1\} \\
& \{1\} & \{0, 1, 2\} \\
& \{0, 1, 2\} & \{1, 3\} \\
& \{1, 3\} & \phi \\
& \phi & \{0, 1, 2\}
\end{array}
\]
```
2.9. Suppose $\mathcal{M} = (Q, \Sigma, q_0, \delta, F)$ is an NFA accepting language $A \subseteq \Sigma^*$. Let $\mathcal{N} = (Q, \Sigma, q_0, \delta, F^c)$, where $F^c = Q - F$ is the complement of $F$. Does machine $\mathcal{N}$ accepts $A^c = \Sigma^* - A$?

Answer: No, not necessarily. Here is a counterexample. Consider $\mathcal{M}$ be given by the sketch to the left below, and the associated machine $\mathcal{N}$ on the right, constructed by swapping the accepting and nonaccepting states of $\mathcal{M}$.

$\mathcal{M}$ accepts language $A = \{w : w = \epsilon \text{ or } w \text{ contains all } a\text{'s or } w \text{ contains all } b\text{'s}\}$. Machine $\mathcal{N}$ accepts only the empty string, which is not $A^c$.

However, if $\mathcal{M}$ were a DFA accepting a language $A$, then its counterpart $\mathcal{N}$, obtaining by swapping accepting and nonaccepting states, would accept $A^c$ because all strings in $A^c$ must terminate in nonaccepting states in $\mathcal{M}$.

2.11. Suppose $A$ and $B$ are two regular languages over the same alphabet. Then $A - B = A \cap B^c$ is a regular language.

Proof: From problem 2.9 above, we know that $A$ regular implies $A^c$ regular. That is, we can transform a DFA accepting $A$ into a DFA accepting $A^c$ by swapping the accepting and nonaccepting states. That is, regular languages are closed under complementation.

Also, regular languages are closed under unions. Therefore $A - B = A \cap B^c = (A^c \cup B)^c$ is a regular language.

2.12. For each of the following regular expressions, give two strings that are in the corresponding language and two strings that are not. The alphabet in each case is $\Sigma = \{a, b\}$.

1. $(a(ba))^b$: the corresponding language contains $ab$ and $abab$. It does not contain $b$ or $a$.
2. $(a \cup b)^*a(a \cup b)^*b(a \cup b)^*a(a \cup b)^*$: the corresponding language contains $aba$ and $aaba$. It does not contain $ab$ or $ba$.
3. $(a \cup ba \cup bb)(a \cup b)^*$: the corresponding language contains $a$ and $ba$. It does not contain $\epsilon$ or $b$.

2.13. Give regular expressions for each of the following languages. The common alphabet is $\Sigma = \{0,1\}$.

1. $\{w : w \text{ contains at least 3 ones}\}$: $1(0 \cup 1)^*1(0 \cup 1)^*1(0 \cup 1)^*1(0 \cup 1)^*$.
2. $\{w : w \text{ contains at least 2 ones and at most 1 zero}\}$: $111^* \cup 0111^* \cup 11^*011^* \cup 111^*$.
3. $\{w : w \text{ contains an even number of zeros and exactly two ones}\}$: $0(00)^*1(00)^*1(00)^* \cup 0(00)^*1(00)^* \cup 0(00)^*1(00)^* \cup 0(00)^*1(00)^*1(00)^* \cup 0(00)^*1(00)^*1(00)^*$.
4. $\{w : w \text{ contains exactly two zeros and at least two ones}\}$: $111^*1^*01^* \cup 1^*01^*1^*1 \cup 1^*011^*1^*1 \cup 1^*011^*01^* \cup 1^*011^*01^* \cup 1^*011^*011^*$. The six options are
   - the two required ones appear before the two zeros
   - the two required ones appear between the two zeros
   - the two required ones appear after the two zeros
   - a required one appears before the first zero and another between the two zeros
   - a required one appears between the two zeros and another after the zeros
   - a required one appears before the two zeros and another after the two zeros.
5. $\{w : w \text{ contains an even number of zeros and each zero is followed by at least one 1}\}$: $1^*((011^*)(011^*))^*$
6. $\{w : \text{ every odd position in } w \text{ is a one}\}$: $(1(0 \cup 1))^*$
2.14.1. Convert \((0 \cup 1)^*000(0 \cup 1)^*\) to an NFA.

2.15. Convert the following DFA to a regular expression. By inspection, we have

\[(a \cup bb^*aa^*b)^*\].

Let’s try to solve via language equations.

\[
\begin{align*}
L_1 &= aL_1 \cup bL_2 \cup \epsilon \\
L_2 &= bL_2 \cup aL_3 \Rightarrow L_2 = b^*aL_3 \\
L_3 &= bL_1 \cup aL_3 \Rightarrow L_3 = a^*bL_1 \\
L_2 &= b^*aa^*bL_1 \\
L_1 &= aL_1 \cup bb^*aa^*bL_1 \cup \epsilon = (a \cup bb^*aa^*b)L_1 \cup \epsilon \\
L_1 &= (a \cup bb^*aa^*b)^* \epsilon = (a \cup bb^*aa^*b)^*. 
\end{align*}
\]
2.16 Convert the following DFA to a regular expression.

\[
L_1 = aL_2 \cup bL_2 \cup \epsilon = (a \cup b)L_2 \cup \epsilon \\
L_2 = aL_2 \cup bL_3 \\
L_3 = aL_1 \cup bL_2 \cup \epsilon
\]

\[
L_2 = aL_2 \cup b(aL_1 \cup bL_2 \cup \epsilon) = (a \cup b)L_2 \cup baL_1 \cup b \\
L_3 = (a \cup b)(a \cup bb)^*baL_1 \cup (a \cup b)(a \cup bb)^*b \cup \epsilon \\
L_1 = ((a \cup b)(a \cup bb)^*ba)^*(((a \cup b)(a \cup bb)^*b \cup \epsilon).
\]

2.21 Prove that the language \( L = \{a^m b^n : m \geq 0, n \geq 0, m \neq n \} \) over \( \Sigma = \{a, b\} \) is not regular.

Proof: Define language \( C = \{a^p b^q : p \geq 0, q \geq 0\} \), which is accepted by the following two-state NFA.

\[ a \longrightarrow 1 \quad \epsilon \longrightarrow 2 \]

Therefore language \( C \) is regular. Also, the language \( B = \{a^n b^n : n \geq 0\} \) is not regular. This language was proved to be not regular in the text (p. 69) via the pumping lemma.

We note that \( B = C - L \), and regular languages are closed under set differences. Consequently, if \( L \) is regular, then \( C - L = B \) is regular — a contradiction.

We conclude that \( L \) is not regular.

2.22 Give counter-examples verifying the following two statements.

1. A subset of a non-regular language is not necessarily non-regular.
   Let \( B = \{0^n 1^n : n \geq 0\} \), which is known to be non-regular. The empty language \( \phi \) is regular, and \( \phi \subset B \).

2. A subset of a regular language is not necessarily regular.
   Again, let \( B = \{0^n 1^n : n \geq 0\} \), which is known to be non-regular. The language \( \{0,1\}^* \) is the set of all strings over alphabet \( \{0,1\} \), and it is clearly regular because it is accepted by a single (accepting) state that cycles on 0 and 1. However, \( B \subset \{0,1\}^* \).