Problem 1. (a) An NFA with a single transition accepting $L = \{aa, bb\}^*$:

![NFA Diagram](image)

(b) Convert the NFA of question (a) to a DFA.

When we apply the subset construction, we get:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>{0}</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>{1}</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>{2}</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>{0, 3}</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>\emptyset</td>
<td>E</td>
</tr>
</tbody>
</table>

The final states are $A$ and $D$ and the start state is $A$. 
Problem 2. (a) Given the incompletely defined DFA $D = (Q, \Sigma, \delta, q_0, F)$, we construct the NFA $N = (Q, \Sigma, \tilde{\delta}, q_0, F)$ whose transition function $\tilde{\delta}$ is defined as follows: for all $p \in Q$ and all $a \in \Sigma$,

$$\tilde{\delta}(p, a) = \begin{cases} \{q\} & \text{if } \delta(p, a) = q \\ \emptyset & \text{if } \delta(p, a) = \emptyset. \end{cases}$$

When we apply the subset construction to $D$ (that is, $N$) since the transition function $\tilde{\delta}$ of $N$ has the property that either $\tilde{\delta}(p, a) = \{q\}$ or $\tilde{\delta}(p, a) = \emptyset$, the only subsets that will be created are of the form $\{q\}$ where $q \in Q$ is reachable from $q_0$, or $\emptyset$, which is a trap state that we denote by $T$. Therefore, the set of states of $D'$ is

$$\{\{p\} \mid p \in Q_r\} \cup \{T\},$$

and the transition function $\delta'$ of the DFA $D'$ is given by

$$\delta'({\{p}\}, a) = \begin{cases} \{q\} & \text{if } \tilde{\delta}(p, a) = \{q\} \\ T & \text{if } \tilde{\delta}(p, a) = \emptyset \end{cases}$$

$$\delta'(T, a) = T,$$

which by the definition of $\tilde{\delta}$ is equivalent to

$$\delta'({\{p}\}, a) = \begin{cases} \{q\} & \text{if } \delta(p, a) = q \\ T & \text{if } \delta(p, a) = \emptyset \end{cases}$$

$$\delta'(T, a) = T,$$

for all $p \in Q_r$ and all $a \in \Sigma$, as claimed.
(b) It is false that a DFA accepts a finite language iff its contains no underlying cycle. This is because, given any DFA, there must be a transition from every state on every input and as a DFA is finite, every DFA has a cycle! For example, the following DFA over the alphabet \{a\} only accepts \(\epsilon\), yet it has a cycle:

![DFA for {\epsilon}](image)

Problem 3.

Let \(\Sigma = \{a_1, \ldots, a_m\}\) be an alphabet. Recall that the sets of languages \(R(\Sigma)_n\) are defined inductively as

\[
R(\Sigma)_0 = \{\{a_1\}, \ldots, \{a_m\}, \emptyset, \{\epsilon\}\},
\]

\[
R(\Sigma)_{n+1} = R(\Sigma)_n \cup \{L_1 \cup L_2, L_1 L_2, L^* \mid L_1, L_2, L \in R(\Sigma)_n\}.
\]

and that the set of regular expressions \(\mathcal{R}(\Sigma)_n\) are defined inductively as

\[
\mathcal{R}(\Sigma)_0 = \{a_1, \ldots, a_m, \emptyset, \epsilon\},
\]

\[
\mathcal{R}(\Sigma)_{n+1} = \mathcal{R}(\Sigma)_n \cup \{(R_1 + R_2), (R_1 \cdot R_2), R^* \mid R_1, R_2, R \in \mathcal{R}(\Sigma)_n\}.
\]

The function \(\mathcal{L}: \mathcal{R}(\Sigma) \to R(\Sigma)\) is defined recursively by

\[
\mathcal{L}[a_i] = \{a_i\},
\]

\[
\mathcal{L}[\emptyset] = \emptyset,
\]

\[
\mathcal{L}[\epsilon] = \{\epsilon\},
\]

\[
\mathcal{L}[(R_1 + R_2)] = \mathcal{L}[R_1] \cup \mathcal{L}[R_2],
\]

\[
\mathcal{L}[(R_1 \cdot R_2)] = \mathcal{L}[R_1] \mathcal{L}[R_2],
\]

\[
\mathcal{L}[R^*] = \mathcal{L}[R]^*.
\]

We prove by induction on \(n \geq 0\) that for every language \(L \in R(\Sigma)_n\) there is some regular expression \(R \in \mathcal{R}(\Sigma)_n\) such that \(\mathcal{L}[R] = L\).

For the base case \(n = 0\), since \(R(\Sigma)_0 = \{\{a_1\}, \ldots, \{a_m\}, \emptyset, \{\epsilon\}\}\) and \(\mathcal{R}(\Sigma)_0 = \{a_1, \ldots, a_m, \emptyset, \epsilon\}\), the claim is true by definition of \(\mathcal{L}\).

For the induction step, pick any language \(L \in R(\Sigma)_{n+1}\). By the definition of \(R(\Sigma)_{n+1}\), either \(L \in R(\Sigma)_n\) or there are languages \(L_1, L_2, L_3 \in R(\Sigma)_n\) such that either \(L = L_1 \cup L_2\) or \(L = L_1 L_2\) or \(L = L_3^*\).
If \( L \in R(\Sigma)_n \), then by the induction hypothesis there is some regular expression \( R \in \mathcal{R}(\Sigma) \) such that \( L = \mathcal{L}[R] \).

Otherwise, by the induction hypothesis there are some regular expressions \( R_1, R_2, R_3 \in \mathcal{R}(\Sigma)_n \) such that \( L_1 = \mathcal{L}[R_1] \), \( L_2 = \mathcal{L}[R_2] \), and \( L_3 = \mathcal{L}[R_3] \). By the recursive definition of \( \mathcal{L} \), we have

\[
\mathcal{L}[(R_1 + R_2)] = \mathcal{L}[R_1] \cup \mathcal{L}[R_2] = L_1 \cup L_2 = L \\
\mathcal{L}[(R_1R_2)] = \mathcal{L}[R_1]\mathcal{L}[R_2] = L_1L_2 = L \\
\mathcal{L}[(R_3^*)] = (\mathcal{L}[R_3]^*)^* = L_3^* = L,
\]

establishing the induction hypothesis. Therefore, \( \mathcal{L} : \mathcal{R}(\Sigma)_n \to R(\Sigma)_n \) is surjective. Since

\[
R(\Sigma) = \bigcup_{n \geq 0} R(\Sigma)_n \quad \text{and} \quad \mathcal{R}(\Sigma) = \bigcup_{n \geq 0} \mathcal{R}(\Sigma)_n,
\]

every \( L \in R(\Sigma) \) belongs to \( R(\Sigma)_n \) for some \( n \geq 0 \), by the surjectivity of \( \mathcal{L} : \mathcal{R}(\Sigma)_n \to R(\Sigma)_n \) there is some \( R \in \mathcal{R}(\Sigma)_n \subseteq \mathcal{R}(\Sigma) \) such that \( \mathcal{L}[R] = L \), so the map \( \mathcal{L} : \mathcal{R}(\Sigma) \to R(\Sigma) \) is also surjective.

**Problem 4.** By definition, \( L^R = \{w^R \mid w \in L\} \). We have

\[
w \in (L_1 \cup L_2)^R \quad \text{iff} \quad w^R \in L_1 \cup L_2 \\
\quad \text{iff} \quad w^R \in L_1 \quad \text{or} \quad w^R \in L_2 \\
\quad \text{iff} \quad w \in L_1^R \quad \text{or} \quad w \in L_2^R \\
\quad \text{iff} \quad w \in L_1^R \cup L_2^R,
\]

which proves that

\[
(L_1 \cup L_2)^R = L_1^R \cup L_2^R.
\]

Recall that it was proved that

\[
(uv)^R = v^Ru^R \quad \text{and} \quad (w^R)^R = w,
\]

for all \( u, v, w \in \Sigma^* \). We have

\[
w \in (L_1L_2)^R \quad \text{iff} \quad w^R \in L_1L_2 \\
\quad \text{iff} \quad (\exists u \in L_1)(\exists v \in L_2)(w^R = uv) \\
\quad \text{iff} \quad (\exists u \in L_1)(\exists v \in L_2)(w = v^Ru^R) \\
\quad \text{iff} \quad (\exists x \in L_1^R)(\exists y \in L_2^R)(w = yx) \\
\quad \text{iff} \quad w \in L_1^R \cup L_2^R,
\]

which proves that

\[
(L_1L_2)^R = L_1^R \cup L_2^R.
\]
We claim that 
\[(L^n)^R = (L^n)^R, \quad \text{for all } n \geq 0.\]
This is proved by induction. For \(n = 0\), we have
\[(L^0)^R = \{\epsilon\}^R = \{\epsilon\} = (L^R)^0,\]
so the base case holds.

Assume the induction hypothesis holds for any \(n \geq 0\). Using \((L^1L_2)^R = L_2^R L_1^R\), we get
\[(L^{n+1})^R = (L^nL)^R = L^R(L^n)^R = L^R(L^R)^n = (L^R)^{n+1},\]
establishing the induction step.

Then, we get
\[(L^*)^R = \left( \bigcup_{n \geq 0} L^n \right)^R = \bigcup_{n \geq 0} (L^n)^R = \bigcup_{n \geq 0} (L^R)^n = (L^R)^*,\]
so
\[(L^*)^R = (L^R)^*,\]
as claimed.

**Problem 5.** Let \(\Sigma = \{a, b\}\).

(a) A DFA accepting
\[L_1 = \{w \in \Sigma^* \mid w \text{ contains an even number of } a's\}\]

![DFA for L1](image)

Figure 4: DFA for \(L_1\)

(b) A DFA accepting
\[L_2 = \{w \in \Sigma^* \mid w \text{ contains a number of } b's \text{ divisible by } 3\}\]
(c) A DFA accepting $L_3 = L_1 \cap L_2$.

The cross-product construction (for intersection) yields:

<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(0, A)$</td>
<td>$(1, A)$</td>
<td>$(0, B)$</td>
</tr>
<tr>
<td>$(0, B)$</td>
<td>$(1, B)$</td>
<td>$(0, C)$</td>
</tr>
<tr>
<td>$(0, C)$</td>
<td>$(1, C)$</td>
<td>$(0, A)$</td>
</tr>
<tr>
<td>$(1, A)$</td>
<td>$(0, A)$</td>
<td>$(1, B)$</td>
</tr>
<tr>
<td>$(1, B)$</td>
<td>$(0, B)$</td>
<td>$(1, C)$</td>
</tr>
<tr>
<td>$(1, C)$</td>
<td>$(0, C)$</td>
<td>$(1, A)$</td>
</tr>
</tbody>
</table>

The start state $(0, A)$ is also the only final state.

**Problem 6.** Let $D_1$ and $D_2$ be two DFA’s over the same alphabet, $\Sigma$. Using the hint, we have
\[ L(D_1) \neq L(D_2) \text{ iff } L(D_1) - L(D_2) \neq \emptyset \text{ or } L(D_2) - L(D_1) \neq \emptyset. \]

However, the cross-product construction yields a DFA accepting \( L(D_1) - L(D_2) \) (or \( L(D_2) - L(D_1) \)) so all we have to do is to come up with a method for testing whether some arbitrary DFA \( D \) accepts a nonempty language. But, we have an algorithm for computing the set \( Q_r \) of states reachable from the start state \( q_0 \) of \( D \), and observe that
\[ L(D) \neq \emptyset \text{ iff } F \cap Q_r \neq \emptyset, \]
and the latter condition can obviously be tested (\( F \) is the set of final states of \( D \)).