Homework 1

Problem 1.
(a) To accept digit strings that contain 481:
\[ Q = \{ \lambda, 4, 48, 481 \}, \Sigma = \{ 0,1,\cdots,9 \}, q_0 = \lambda, A = \{ 481 \}. \]

To define \( \delta \), we use \( \bar{a} \) for all letters (well, digits) other than \( a \).
\[ \delta(\lambda, \bar{a}) = \delta(4, \bar{a}) = \delta(48, \bar{a}) = \delta(481, \bar{a}) = \lambda \]
\[ \delta(\lambda, 4) = 4, \delta(4, 8) = 48, \delta(48, 1) = \delta(481, anything) = 481 \]

(b) to accept strings of \( a \)'s of length divisible by 2 or 7:
\[ Q = \{ (i, j) : 0 \leq i < 2 \text{ and } 0 \leq j < 7 \}, \Sigma = \{ a \}, q_0 = (0,0), A = \{ (i, j) : ij = 0 \} \]
\[ \delta((i, j), a) = (i + 1 \mod 2, j + 1 \mod 7). \]

(c) accept if the 0-content is divisible by 2 and 1-content is divisible by 3 (I am almost cutting and pasting the previous solution):
\[ Q = \{ (i, j) : 0 \leq i < 2 \text{ and } 0 \leq j < 3 \}, \Sigma = \{ 0,1 \}, q_0 = (0,0), A = \{ (i, j) : ij = 0 \} \]
\[ \delta((i, j), 0) = (i + 1 \mod 2, j), \delta((i, j), 1) = (i, 3 + 1 \mod 3). \]
accept if \( bbb \) occurred thrice, overlapping occurrences permitted:
\[ Q = \{ 0,1,2 \} \times \{ \lambda, b, bb, bbb \} \cup \{ 3 \}, \Sigma = \{ a, b \}, q_0 = (0,\lambda), A = \{ 3 \} \]
\[ \delta((i, anything), a) = (i, \lambda), \delta(3, a) = 3., \]
\[ \delta((i, \lambda), b) = (i, b), \]
\[ \delta((i, b), b) = (i, bb), \]
\[ \delta((i, bb), b) = (i + 1, bbb) \text{ for } i < 2, \]
\[ \delta((2, bb), b) = 3. \]

(e) accept ternary representation of numbers not divisible by 4:
\[ Q = \{ 0,1,2,3 \}, \Sigma = \{ 0,1,3 \}, q_0 = 0, A = \{ 1,2,3 \}, \delta(i, j) = 3i + j \mod 3. \]

Problem 2. Product automaton will have \( Q = \{ 1,2 \} \times \{ 1,2,3 \}, \Sigma = \{ a, b \}, q_0 = (1,1) \) and \( \delta \) defined by the following table:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1)</td>
<td>(1,2)</td>
<td>(2,3)</td>
</tr>
<tr>
<td>(1,2)</td>
<td>(1,3)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(1,3)</td>
<td>(1,1)</td>
<td>(2,2)</td>
</tr>
<tr>
<td>(2,1)</td>
<td>(2,2)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>(2,2)</td>
<td>(2,3)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(2,3)</td>
<td>(2,1)</td>
<td>(1,2)</td>
</tr>
</tbody>
</table>

(a) For the intersection, \( A = \{ (2,1) \} \).
(b) For the union, \( A = Q - \{ 1 \} \times \{ 2 \} \).
**Problem 3.** The basis, for \( y = \lambda \), holds by the definition of \( \hat{\delta} \):
\[
\hat{\delta}(\hat{\delta}(q, x), \lambda) = \hat{\delta}(q, x) = \hat{\delta}(q, x\lambda)
\]
Inductive step: true for \( y \), we prove for \( ya \),
\[
\hat{\delta}(\hat{\delta}(q, x), ya) = \text{[definition]}
\]
\[
\hat{\delta}(\hat{\delta}(q, x), y, a) = \text{[ind. hyp. ]}
\]
\[
\hat{\delta}(\hat{\delta}(q, xy), a) = \text{[definition again]}
\]
\[
\hat{\delta}(q, xy\alpha) = \text{[qed]}
\]

**Problem 4.** We almost repeat the solution of 1e:
\[
Q = \{0, \cdots, k - 2\}, \Sigma = \{0, \cdots, p - 1\}, q_0 = 0, A = \{0\}, \delta(i, j) = pi + j \mod k.
\]

**Problem 6,** page 317. We need a language, a succinct NFA and a proof that DFA must have many states.
Language: \((0 + 1)^* 0(0 + 1)^{n-1}\).
Succinct NFA: \(Q = \{0, 1, \ldots, n\}, \Sigma = \{0, 1\}, q_0 = 0, A = \{n\}, \delta(0, 1) = \{0\}, \delta(0, 0) = \{0, 1\}, \delta(i, a) = \{i + 1\} \text{ if } 0 < i < n, \delta(n, a) = \text{empty set.}

Why it works: we accept if the suffix of the input of length \( n \) starts with 0, the machine guesses when the suffix starts, and after the guess it counts how many letters of the suffix it has read already. So state 0 is before guessing and other states are after the guess.

Why we do not have a succinct deterministic automaton. The intuition: if the automaton outputs 0 when it accepts and 1 when it rejects, then after a while the output equals to the input, but with the delay of \( n - 1 \) steps. Thus if the machine is in some state \( q \), and we supply it with any kind of input, say, 000..., both the current output together the next \( n - 1 \) output symbols form the suffix of the input so far of length \( n \), moreover, this output is determined solely by the current state as it does not depend on the input. Thus we must have as many states as we have words of length \( n \), which is \( 2^n \).
Homework 2

Problem 1. First I convert the automaton to my favorite form.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>0</td>
<td>{0,1}</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>{2}</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>{3}</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>{}</td>
</tr>
</tbody>
</table>

Now we calculate possible values of $\hat{\delta}(0, w)$:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>{0}</td>
<td>{0,1}</td>
</tr>
<tr>
<td></td>
<td>{0,1}</td>
<td>{0,1,2}</td>
</tr>
<tr>
<td></td>
<td>{0,1,2}</td>
<td>{0,1,2,3}</td>
</tr>
<tr>
<td>A</td>
<td>{0,1,2,3}</td>
<td>{0,1,2,3}</td>
</tr>
</tbody>
</table>

Problem 2. By induction on $|y|$, $\text{rev } xy = \text{rev } y \text{ rev } x$. For $|y| \leq 2$ it follows from definition, for $y' = ya$ we have

$$\text{rev } xya = a \text{ rev } xy = a \text{ rev } y \text{ rev } x = \text{rev } ya \text{ rev } x$$

From that we can conclude that $\text{rev } KL = \text{rev } L \text{ rev } K$ and $\text{rev } K^* = (\text{rev } K)^*$, while it is obvious that $\text{rev } (K \cup L) = \text{rev } K \cup \text{rev } L$.

Using these observation as conversion rules, one can convert a regular expression for language $L$ into a regular expression for $\text{rev } L$.

Problem 3. We can use closure properties we already know. Define a substitution $s(0) = \{0, \hat{0}\}$ and $s(1) = \{1, \hat{1}\}$. If $L$ is regular, so is $L_1 = s(L)$.

Now we define

$$L_2 = L_1 \cap (0 + 1)^*(\hat{0} + \hat{1} + \varepsilon)(0 + 1)^*(\hat{0} + \hat{1} + \varepsilon)(0 + 1)^*$$

(3)

$L_2$ is almost like $L$, except that we can change at most 2 letters by putting hats on them.

Now we define substitution $t(0) = \{0\}$, $t(1) = \{1\}$, $t(\hat{0}) = t(\hat{1}) = \{0,1\}$ and $L_3 = t(L_2)$.

As a result, $L_3$ is like $L$, except that we can change at most 2 letters, so $L_3 = N_2(L)$. By changing the regular expression used in (3) we can show the same for any $N_k(L)$. 

Homework 3

Problem 1.
(a) $b^* (ab^* ab^*)^*$
(b) $a^* (ba^* ba^*)^* ba^*$
(c) $b^* (ab^* ab^*)^* + a^* (ba^* ba^*)^* ba^*$
(d) $(aa + (ab + ba) (aa + bb) (ab + ba)^* (ab + ba) + bb)^* (a(bb)^* ba + b)(a(bb)^* a)^*$

Problem 2.
(a) $L_0 = (000^* + 111^*)^*$. We will construct a quotient machine.

Thus we have a 6-state automaton defined by this table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow A$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) $L_0 = (01 + 10)(01 + 10)(01 + 10)^*$, again we make a quotient machine.

Thus we have a 11-state automaton defined by this table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow A$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
(c) \( L_0 = (0 + 1(01^*0)^*)^* \), quotient machine once more:

\[ L_0 \setminus 0 = L_0 \]
\[ L_0 \setminus 1 = (01^*0)^*L_0 = L_1 \]
\[ L_1 \setminus 0 = 1^*0(01^*0)^*1L_0 = L_2 \]

Thus we have a 3-state automaton defined by this table:

<table>
<thead>
<tr>
<th>A,→</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Problem 2.** The lost language of the Middle Earth can be recovered using MiddleThirds operation. Aragorn can do it, can you?

Suppose that \( L = L(<Q, \Sigma, q_0, \delta, A>) \).

We will use composition of binary relations of type \( R \subset Q \times Q \). In particular, \( p(R \circ S)r \) if and only if for some \( q \) we have \( pRq \) and \( qSr \). We need the following relations:

- \( pIq \) if and only if \( p = q \)
- \( pR_aq \) if and only if \( q \in \delta(p, a) \)
- \( pRq \) if and only if \( pR_aq \) for some \( a \in \Sigma \).

We can now define \( R_u \) by induction: \( R_\varepsilon = I \) and \( R_{u^a} = R_u \circ R_a \). Clearly, \( u \in L \) if and only if \( q_0R_up \) for some \( p \in A \).

One can see that \( u \) belongs to MiddleThirds \( L \) if and only if \( q_0(R^{[u]} \circ R_u \circ R^{[u]})p \) for some \( p \in A \).

Note that given \( R^{[u]} \), \( R_u \) and \( a \) we can compute \( R^{[u^a]} = R^{[u]} \circ R \) and \( R_{u^a} = R_u \circ R_a \). Moreover, all these relations are finite objects. Thus we can define a DFA for Middle Thirds \( L \):

- \( Q' = \{(R, S) : R, S \subset Q \times Q\} \)
- \( q_0' = (I, I) \)
- \( A' = \{(R, S) : q_0(R \circ S \circ R)p \text{ for some } p \in Q\} \)
- \( \delta'((R, S), a) = (R \circ R, S \circ R_a) \)
Problem 1. If we replace in grammar $G$ each production $A \to \alpha$ with $A \to \alpha R$, then $L(G^R) = L(G)^R$. Note that $L$ is regular if and only if $L^R$ is regular, and if $G$ is (strongly) left-linear then $G^R$ is (strongly) right-linear. Thus it suffices to prove the claim for (strongly) right-linear grammars only.

We can convert a right linear grammar to form $V \to TV + T + \varepsilon$ with a method used in obtaining Chomsky normal form, which means that if $x \neq \varepsilon$ then the production $A \to ax B$ can be replaced with two productions $A \to aA'$ and $A' \to xB$, where $A'$ is a new variable.

We can also eliminate $\varepsilon$-productions with a method used in obtaining Chomsky normal form, so if there is production $A \to \varepsilon$ we add production $B \to a$ for every production $B \to aA$ and then we eliminate the $\varepsilon$ production. The new language can differ from the old one by not containing $\varepsilon$ word, so it is regular if and only if the old language is regular. Thus it suffices to prove the claim for right-linear grammars only (and with exactly one terminal symbol in each production right-hand-side).

Given such a grammar $(V, T, S, P)$ we can define NFA $(Q = V \cup \{A_{\text{accept}}\}, \Sigma = T, q_0 = S, \delta, F = \{A_{\text{accept}}\})$ where $B \in \delta(A, a)$ if either $A \to aB$ is a production or $B = A_{\text{accept}}$ and $A \to a$ is a production.

It is easy to show by induction on $|X|$ that for $A \in V$ we have $A \in \hat{\delta}(S, x)$ if and only if $S \xrightarrow{*} xA$, and that $A_{\text{accept}} \in \hat{\delta}(S, x)$ if and only if $S \xrightarrow{*} x$.

Problem 2. We can prove it by induction on $|x|$. We assume that $x$ has the same number of $a$’s and $b$’s. If $x = \varepsilon$ then $S \to x$. Otherwise $x$ must have some $k$ $a$’s and $k$ $b$’s, and the total length of $2k$ for some $k > 0$.

If $x = ayb$ then $y$ has $k - 1$ $a$’s and $b$’s, so by inductive assumption $S \to aSb \xrightarrow{*} ayb$.

If $x = aya$ then $y$ has $k - 2$ $a$’s and $k$ $b$’s. Consider $y = vw$ where $v$ is the shortest prefix of $y$ that has more $b$’s than $a$’s. Clearly, $v$ has one more $b$ than $a$’s, otherwise we could make $v$ shorter by one symbol. This implies that $av$ has the same number of $a$’ and $b$’, and consequently the same holds for $wa$ and thus $S \to SS \xrightarrow{*} avS \xrightarrow{*} avwa$.

The cases of $x = bya$ and $x = byb$ are similar.

Problem 3. The following grammar follows exactly the inductive definition:

$$S \to \varepsilon \mid (S) \mid [S] \mid SS.$$ 

Problem 4. We will have just one state so we skip the state in description of transitions and the stack alphabet $\{S, \) \}$. 

$$\delta(\varepsilon, S) = \{\varepsilon\},$$

$$\delta(\varepsilon, \) = \{\varepsilon\},$$

$$\delta(\varepsilon, \) = \{ S \},$$

$$\delta(\varepsilon, S) = \{\varepsilon\}.$$
Homework 6, page 307

Problem 1. The grammar is $S \rightarrow aB \mid bA$, $A \rightarrow aS \mid a \mid bAA$, $B \rightarrow bS \mid b \mid aBB$ We will show by induction on $|u|$ that

a) $A \overset{*}{\Rightarrow} u$ if and only if $a(u) = b(u) + 1$ ($u$ has one more $a$ than $b$’s);

b) $B \overset{*}{\Rightarrow} u$ if and only if $b(u) = a(u) + 1$;

c) $S \overset{*}{\Rightarrow} u$ if and only if $a(u) = b(u)$.

Basis: for a word of length 1 and 2, by simple inspection of the rules. Note that $X \overset{*}{\Rightarrow} u$ implies $|X| \geq 1$.

Inductive step:

a word $u = av$ of length $n > 1$ has the same number of $a$’s and $b$’s if and only if $v$ has one more $b$, hence if and only if $S \rightarrow aB \overset{*}{\Rightarrow} av$;

the case of a word $u = bv$ of length $n > 1$ with the same number of $a$’s and $b$’s is symmetric;

a word $u = av$ of length $n > 1$ has one more $a$ than $b$’s if and only if $v$ has the same number of $a$’s and $b$’s if and only if $A \rightarrow aS \overset{*}{\Rightarrow} av$;

a word $u = bv$ of length $n > 1$ has one more $a$ than $b$’s if and only if $v$ has two more $a$’s than $b$’s; let $v = wx$ where $w$ is the shortest prefix of $v$ with more $a$’s than $b$’s; $w$ has one more $a$, hence $x$ also has one more $a$, hence we can derive $u = bv = bwx$ from $bAA$;

we have two more symmetric cases.

Problem 2. We again convert a grammar to a 1-state PDA, so we omit references to the state in describing the transitions; starting symbol $S$.

$\delta(a, S) = \{B\}$

$\delta(b, S) = \{A\}$

$\delta(a, A) = \{\varepsilon, S\}$

$\delta(b, A) = \{AA\}$

$\delta(b, B) = \{\varepsilon, S\}$

$\delta(a, B) = \{BB\}$
Problem 3. \( L = \{ b \$ c : \exists k \text{ s.t. } b \text{ is a binary code of } k \text{ and } c \text{ of } k+1 \} \)

a) We will use pumping lemma, and for \( n \) we give word \( u = 1^n 0^{n+1} 1 0^n 1 = vwxyz \).

Suppose that \( wxy \) does not contain $, then deleting \( w \) and \( y \) (pumping down) alters the code of a number on one side of $ only, so the result is not in \( L \).

Suppose that \( w \) or \( v \) does contain $, pumping down deletes $, so the result is not in \( L \).

It remains to consider the case when \( x \) contains $, \( w \) is contained in \( 0^{n+1} \) and \( y \) is contained in \( 1^n \) that follows $. Becuase \( |wy| < n \), after pumping down the code on the left side of $ ends with 0, so the code on the right has the same length, so we have \( |w| = |y| \). It is easy to check that after pumping down the code on the left side of $ cannot be equal to that on the right (plus 1).

b) \( L = \{ 0u\$u^R 1 : u \in (0+1)^* \} \cup \{ 1^k\$10^k : k > 0 \} \) It is easy to find a CFG for \( L \) once we define it in that way.

Problem 4. We had the following grammar: \( S \rightarrow \varepsilon | (S) | [S] | SS \).

We can modify it as follows: if we do not generate \( \varepsilon \) it is safe to start from rule \( S \rightarrow SS \) because we can remove the second \( S \) using \( S \rightarrow \varepsilon \). Then it is safe to apply one of the two rules that generate parenthesis, because before applying a rule like that, a leftmost derivation generates some \( S^k \) and the final \( S^{k-1} \) can be generated from our second \( S \). Thus it is safe to start with \( S \rightarrow (S)S \) or \( S \rightarrow [S]S \). Thus our grammar can be

\[
S \rightarrow \varepsilon | (S)S | [S]S.
\]

After elimination of \( \varepsilon \)-production it becomes

\[
S \rightarrow (S)S | [S]S | () | [S]S | [] | [S] | []
\]

This is pretty much in Greibach form. We can introduce symbols \( A \) for \( ) \) and \( B \) for \( ] \) and then this becomes

\[
S \rightarrow (SA) | (A | [SB] | [B[A)
\]

Converting the last grammar to Chomsky’s form is easy. The grammar above is correct because we have applied transformations to grammars that do not change the language that is generated.