1 Finite State Automata and Regular Languages

Problem 1. Consider the following NFA $A$ over \{a, b\}:

$A$ is intended to recognise the language

$L = \{z\alpha b \mid z \in \{a, b\}, \alpha \in \{a\}^*\}$

of strings over the alphabet \{a, b\} that start with either $a$ or $b$, end with $b$ and have an arbitrary number of $a$’s between the first and the last letter.

(i) The following transition table shows a DFA $A_D'$ that is obtained from $A$ by the method of “lazy” subset construction:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\rightarrow {s_0}$</td>
<td>${s_1, s_2}$</td>
<td>${s_2}$</td>
</tr>
<tr>
<td>${s_1, s_2}$</td>
<td>${s_1, s_2}$</td>
<td>${s_3}$</td>
</tr>
<tr>
<td>${s_2}$</td>
<td>${s_1, s_2}$</td>
<td>${s_3}$</td>
</tr>
<tr>
<td>$\emptyset{s_3}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
For the sake of convenience, let’s define a DFA $A_D$ that is exactly like $A'_D$ but states are renamed. The following table shows $A_D$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_4$</td>
<td>$q_4$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_1$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_4$</td>
<td>$q_4$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_4$</td>
<td>$q_4$</td>
</tr>
</tbody>
</table>

The transition diagram of $A_D$ is as follows:

(ii) The initial split differentiates non-final and final states:

\[
\begin{bmatrix}
q_0, q_1, q_2, q_4, q_3
\end{bmatrix}
\]

$b$ splits $[q_0, q_1, q_2, q_4]$:

\[
\begin{bmatrix}
q_0, q_4, q_1, q_2, q_3
\end{bmatrix}
\]

$a$ (as well as $b$) splits $[q_0, q_4]$:

\[
\begin{bmatrix}
q_0, q_4, q_1, q_2, q_3
\end{bmatrix}
\]

Neither $a$ nor $b$ splits the only remaining non-singleton group $[q_1, q_2]$. Every other group is a singleton. Therefore, the algorithm terminates. States $q_1$ and $q_2$ are equivalent.

(iii) $A_m$ is as follows
(iv) Prove that $A_m$ recognises $L$.

**Proof:** We need to prove the following two statements

(a) for all $w$, if $\delta^*(q_0, w) = q_3$ then $w \in L$
(b) for all $w$, if $w \in L$ then $\delta^*(q_0, w) = q_3$

Each of these statements can be reformulated as follows

(a) for all $w$, if $\delta^*(q_0, w) = q_3$ then either $w = a\alpha b$ or $w = b\alpha b$, where $\alpha \in \{a\}^*$
(b) for all $\alpha$, if $\alpha \in \{a\}^*$ then $\delta^*(q_0, a\alpha b) = q_3$ and $\delta^*(q_0, b\alpha b) = q_3$

Below we prove each of these statements

(a) • $|w| = 0$
Hence, $w = \epsilon$. $\delta^*(q_0, \epsilon) = q_0$. State $q_0$ is a state distinct from $q_3$, therefore the antecedent of the conditional statement is false. Therefore, the whole statement is true.

• $|w| = 1$
Hence, either $w = a$ or $w = b$. In both cases $\delta^*(q_0, w) = q_1$. State $q_1$ is a state distinct from $q_3$, therefore the antecedent of the conditional statement is false. Therefore, the whole statement is true.

• $|w| > 1$
Hence, $w$ is of the form $x\alpha y$, where $x, y \in \{a, b\}$ and $\alpha \in \{a, b\}^*$. Assume $\delta^*(q_0, x\alpha y) = q_3$.
By definition of $\delta^*$ the following holds

$$\delta^*(q_0, x\alpha y) = \delta^*(\delta(q_0, x), \alpha y)$$

$\delta(q_0, x)$ is defined for $x \in \{a, b\}$ as follows:

$$\delta(q_0, a) = q_1$$
$$\delta(q_0, b) = q_1$$

Therefore, regardless whether $x$ is $a$ or $b$, $\delta(q_0, x) = q_1$. Hence,

$$\delta^*(q_0, x\alpha y) = \delta^*(q_1, \alpha y)$$
\[ \delta^*(q_1, x) = \delta(\delta^*(q_1, x), y) = q_3 \]

There is only one one-step transition that leads to \( q_3 \). It is labelled \( b \). Therefore \( y \) must be \( b \) and \( \delta^*(q_1, \alpha) \) must be \( q_1 \).

Consequently, it is sufficient to prove the following statement:

for all \( \alpha \in \{ a, b \}^* \), if \( \delta^*(q_1, \alpha) = q_1 \) then \( \alpha \in \{ a \}^* \)

Lemma 1 proves this statement.

(b) Assume \( \alpha \in \{ a \}^* \). We need to prove that \( \delta^*(q_0, a \alpha b) = q_3 \) and \( \delta^*(q_0, b \alpha b) = q_3 \). We consider \( \delta^*(q_0, a \alpha b) \) and \( \delta^*(q_0, b \alpha b) \) separately:

- \( \delta^*(q_0, a \alpha b) = \delta^*(\delta(q_0, a \alpha b)) = \delta^*(q_1, \alpha b) = \delta(\delta(q_1, \alpha), b) \).
- \( \delta^*(q_0, b \alpha b) = \delta^*(\delta(q_0, b \alpha b)) = \delta^*(q_1, b \alpha b) = \delta(\delta(q_1, \alpha), b) \).

In both cases we reached \( \delta(q_1, \alpha), b) \). By Lemma 2, \( \delta^*(q_1, \alpha) = q_1 \). Therefore, \( \delta^*(q_1, \alpha), b) = \delta(q_1, b) = q_3 \)

Lemma 1

For all \( \beta \in \{ a, b \}^* \), if \( \delta^*(q_1, \beta) = q_1 \) then \( \beta \in \{ a \}^* \)

Proof:

- \( |\beta| = 0 \). \( \beta = \epsilon \) and \( \beta \in \{ a \}^* \).
- \( |\beta| > 0 \). This means that \( \beta = x \alpha \) where \( x \in \{ a, b \} \) and \( \alpha \in \{ a, b \}^* \).

We need to prove the following: if \( \delta^*(q_1, x \alpha) = q_1 \) then \( x \alpha \in \{ a \}^* \)

Assume

\[ \delta^*(q_1, x \alpha) = q_1 \] (1)

By the definition of \( \delta^* \) the following holds:

\[ \delta^*(q_1, x \alpha) = \delta^*(\delta(q_1, x), \alpha) \] (2)

For the sake of contradiction assume that \( x = b \), then

\[ \delta(q_1, x) = \delta(q_1, b) = q_3 \]. Then

\[ \delta^*(q_1, x \alpha) = \delta^*(q_3, \alpha) = q_1 \]

This means that from state \( q_3 \) we reach \( q_3 \) after reading \( \alpha \). But all transitions from \( q_3 \) lead to the “sink” state \( q_4 \). Therefore \( \delta^*(q_3, \alpha) = q_1 \) is impossible. We reached a contradiction. Therefore, \( x \neq b \) and \( x \) can only be equal to \( a \), since \( a \) is the only remaining symbol of the alphabet.

Hence we can rewrite \( \delta^*(\delta(q_1, x), \alpha) \) in (2) as follows:

\[ \delta^*(\delta(q_1, x), \alpha) \]

Since \( \delta(q_1, a) = q_1 \) and by assumption (1) the following holds:

\[ \delta^*(\delta(q_1, a), \alpha) = \delta^*(q_1, a) = q_1 \]

The only one possible way to reach \( q_1 \) from \( q_1 \) is a transition \( \delta(q_1, a) \) or a sequence of them. Therefore \( \alpha \in \{ a \}^* \).

Hence, \( x \alpha \in \{ a \}^* \). \( \square \)
Lemma 2
For all $\alpha \in \{a\}^*$, $\delta^*(q_1, \alpha) = q_1$

Proof: Induction on the length of $\alpha$.

- Base case: $|\alpha| = 0$. $\delta^*(q_1, \epsilon) = q_1$
- Induction: $|\alpha| > 0$.

Induction hypothesis: $\delta^*(q_1, a^n) = q_1$, where $a^n$ is a string consisting of a sequence of $n$ $a$’s. We need to prove $\delta^*(q_1, a^{n+1}) = q_1$.

$\delta^*(q_1, a^{n+1}) = \delta(\delta^*(q_1, a^n), a)$

By induction hypothesis $\delta^*(q_1, a^n) = q_1$, hence

$\delta^*(q_1, a^{n+1}) = \delta(q_1, a) = q_1$

□

Problem 2. Let $L$ be the language

$L = \{w \in \{0, 1\}^* \mid w \in \{0\}^* \text{ or } w \in \{1\}^*\}$

Construct a deterministic finite automaton that accepts all strings that are in $L$, and rejects all strings that are not in $L$.

Problem 4. Let $L$ be the language

$L = \{w \in \{0, 1\}^* \mid w \text{ has a } 1 \text{ in the third position from the right}\}$

Construct a non-deterministic finite automaton that accepts all strings that are in $L$, and rejects all strings that are not in $L$. 
Problem 3. Let $L$ be the language

$$L = \{ w \in \{0, 1\}^* \mid \text{each block of 5 consecutive symbols contains at least two 0's}\}$$

- Construct a **deterministic finite automaton** that accepts all strings that are in $L$, and rejects all strings that are not in $L$.

- Explain in plain English the intuition behind the constructed automaton.

We first define a non-minimal DFA $A_n$ that recognizes $L$. The name of a state conveys the information about the sequence of 0s and 1s of the last 4-symbols block that the automaton has “processed”. Thus, if the sequence of the last four processed symbols was 0010, the automaton is in state named “0010”. Then $\delta(0010, 0) = 0100$ and $\delta(0010, 1) = 0101$. We define the only non-accepting state sink. Whenever the automaton is in a state named with a sequence of four symbols and at least three of these symbols are 1s, the automaton goes to the rejecting state sink on input 1. The only exception is the string 1111, as discussed below.

Since the automaton has to accept all strings of length smaller than 5, there are states for these strings. The only string containing four consecutive 1s that is accepted is the string of length 4. State 1111 is the accepting state corresponding to strings 1111. If the first four 1’s of a string are followed by a symbol, the automaton performs a transition to sink.

The initial state is named “$\epsilon$”. It is the state for the empty string.

Thus the automaton is as follows:
$A_n$ can be minimized. The equivalence classes of states are as follows:

\[
\begin{array}{c|cc}
\epsilon & 0 & 1 \\
\hline
0 & 00 & 01 \\
1 & 10 & 11 \\
00 & 000 & 001 \\
01 & 010 & 011 \\
10 & 100 & 101 \\
11 & 110 & 111 \\
000 & 0000 & 0001 \\
001 & 0010 & 0011 \\
010 & 0100 & 0101 \\
011 & 0110 & 0111 \\
100 & 1000 & 1001 \\
101 & 1010 & 1011 \\
110 & 1100 & 1101 \\
111 & 1110 & 1111 \\
0000 & 0000 & 0001 \\
0001 & 0010 & 0011 \\
0010 & 0100 & 0101 \\
0011 & 0110 & 0111 \\
0100 & 1000 & 1001 \\
0101 & 1010 & 1011 \\
0110 & 1100 & 1101 \\
0111 & 1110 & \text{sink} \\
1000 & 0000 & 0001 \\
1001 & 0010 & 0011 \\
1010 & 0100 & 0101 \\
1011 & 0110 & \text{sink} \\
1100 & 1000 & 1001 \\
1101 & 1010 & \text{sink} \\
1110 & 1100 & \text{sink} \\
1111 & \text{sink} & \text{sink} \\
\end{array}
\]

Among equivalent states, we keep the states that are emphasized in bold above. This gives us the following minimal automaton $A_m$:

---

Assignment 1 — Automata, Languages, and Computability
Here is $A_m$’s transition diagram:

\[
\begin{array}{c|cc}
\gamma & 0 & 1 \\
\hline
\epsilon & 0000 & 1 \\
0 & 1010 & 11 \\
1 & 0110 & 111 \\
11 & 1110 & 1111 \\
111 & 0000 & 0001 \\
000 & 1010 & 0011 \\
001 & 0110 & 0111 \\
011 & 1010 & 1011 \\
010 & 0000 & 1101 \\
110 & 1110 & \text{sink} \\
100 & 0000 & 0101 \\
101 & 0110 & \text{sink} \\
110 & 1010 & \text{sink} \\
111 & 0000 & \text{sink} \\
111 & \text{sink} & \text{sink} \\
\text{sink} & \text{sink} & \text{sink} \\
\end{array}
\]
2 Grammar

Problem 5. From the NFA \( A \) given in Problem 1, derive a right-linear grammar using the algorithm given in the lectures. The grammar is as follows

\[
G = (\{a, b\}, \{S_0, S_1, S_2, S_3\}, S_0, P)
\]

where the set \( P \) consists of the following production rules

\[
\begin{align*}
S_0 & \rightarrow aS_1 \\
S_0 & \rightarrow aS_2 \\
S_0 & \rightarrow bS_2 \\
S_1 & \rightarrow aS_2 \\
S_1 & \rightarrow bS_3 \\
S_2 & \rightarrow aS_1 \\
S_2 & \rightarrow aS_2 \\
S_2 & \rightarrow bS_3 \\
S_3 & \rightarrow \epsilon
\end{align*}
\]

Problem 2.6. Consider the grammar \( G = (\{S, T\}, \{0, 1\}, S, P) \), where \( P \) consists of the following productions

\[
\begin{align*}
S & \rightarrow 0S \mid 1T \mid 0 \\
T & \rightarrow 1T \mid 1
\end{align*}
\]

Show that no string in the language \( L(G) \) contains the substring 10.

Proof: Let \( \alpha \) be a sentential form derived using \( G \). We show by induction on the length \( n \) of the derivation of \( \alpha \) that \( \alpha \) does not contain any of the substrings 10, 1S, S0, SS, T0 or TS as substring.

Base case: \( n = 1 \). The possible derivations of length 1 of sentential forms for \( G \) are \( S \Rightarrow 0S, S \Rightarrow 1T, \) and \( S \Rightarrow 0 \). Neither 0S, nor 1T, nor 0 contain 10, 1S, S0, SS, T0 or TS as substring.

Inductive case: Let \( S \Rightarrow^* \alpha \Rightarrow \beta \) be a derivation of \( \beta \) of length \( n + 1 \). By inductive hypothesis, \( \alpha \) does not contain 10, 1S, S0, SS, T0 or TS as substring.

1. If the last derivation step uses \( S \rightarrow 0S \), since \( \alpha \) does not contain 1S, SS or TS, the derivation step cannot introduce 10, S0 or T0 in \( \beta \).
2. If the last derivation step uses \( S \rightarrow 1T \), since \( \alpha \) does not contain S0 or SS, the derivation step cannot introduce T0 or TS in \( \beta \).
3. If the last derivation step uses \( S \rightarrow 0 \), since \( \alpha \) does not contain 1S, SS or TS, the derivation step cannot introduce 10, S0 or T0 in \( \beta \).
4. If the last derivation step uses \( T \rightarrow 1T \), since \( \alpha \) does not contain T0 or TS, the derivation step cannot introduce T0 or TS in \( \beta \).
5. If the last derivation step uses \( T \rightarrow 1 \), since \( \alpha \) does not contain T0 or TS, the derivation step cannot introduce 10 or 1S in \( \beta \).
3 Context Free Languages and Pushdown Automata

**Problem 3.6.** Show that the language \( \{ uawb \mid u, w \in \{a, b\}^*, \text{with } |u| = |w| \} \) is context free by exhibiting a context free grammar that generates it.

A CFG grammar that generates this language is \( G = (\{a, b\}, \{S, T, X\}, S, P) \), where \( P \) consists of the following productions:

\[
\begin{align*}
S & \rightarrow Tb \\
T & \rightarrow a \mid XTX \\
X & \rightarrow a \mid b
\end{align*}
\]

**Problem 7.** Consider the context-free grammar \( G = (\{S, A, B\}, \{a, b, c\}, P, S) \), where \( P \) consists of the following productions

\[
\begin{align*}
S & \rightarrow aA \\
A & \rightarrow BA \mid a \\
B & \rightarrow bS \mid cS
\end{align*}
\]

Construct a PDA \( M \) that accepts \( L(G) \) by empty stack. Draw the parse tree of \( G \) for the string \( abaaa \), and show the corresponding execution trace for \( M \).

The parse tree for the string \( abaaa \):

\[
\begin{align*}
& S \\
& \quad A \\
& \quad \quad a \\
& \quad B \\
& \quad \quad S \\
& \quad \quad \quad a \\
& \quad \quad b
\end{align*}
\]

\( M = (\{q_0, q_1, q_2\}, q_0, \{q_2\}, \{a, b, c\}, \{a, b, c, S, A, B, Z\}, Z, \delta) \) with \( \delta \) defined as follows:

\[
\begin{align*}
\delta(q_0, \epsilon, Z) &= \{(q_1, SZ)\} \\
\delta(q_1, \epsilon, S) &= \{(q_1, aA)\} \\
\delta(q_1, \epsilon, A) &= \{(q_1, BA), (q_1, a)\} \\
\delta(q_1, \epsilon, B) &= \{(q_1, bS), (q_1, cS)\} \\
\delta(q_1, a, a) &= \{(q_1, \epsilon)\} \\
\delta(q_1, b, b) &= \{(q_1, \epsilon)\} \\
\delta(q_1, c, c) &= \{(q_1, \epsilon)\} \\
\delta(q_1, \epsilon, Z) &= \{(q_2, \epsilon)\}
\end{align*}
\]
The execution trace of \( M \) for the string \( abaaa \) is as follows:

\[
(q_0, abaaa, Z) \vdash (q_1, abaaa, SZ)
\]
\[
\vdash (q_1, baaa, AZ)
\]
\[
\vdash (q_1, baaa, BAZ)
\]
\[
\vdash (q_1, baaa, bSAZ)
\]
\[
\vdash (q_1, a, AZ)
\]
\[
\vdash (q_1, a, aZA)
\]
\[
\vdash (q_1, a, aAZ)
\]
\[
\vdash (q_1, a, aAAZ)
\]
\[
\vdash (q_1, aa, AAZ)
\]
\[
\vdash (q_1, aa, aAZ)
\]
\[
\vdash (q_1, a, AZ)
\]
\[
\vdash (q_1, a, AAZ)
\]
\[
\vdash (q_1, a, aAZ)
\]
\[
\vdash (q_1, ϵ, Z)
\]
\[
\vdash (q_1, ϵ, aAZ)
\]
\[
\vdash (q_1, ϵ, AAZ)
\]
\[
\vdash (q_1, ϵ, aAAZ)
\]
\[
\vdash (q_2, ϵ, ϵ)
\]

4 OPTIONAL: Turing Machine and Computability

**Problem 8.** Design a 2-tape Turing machine accepting the language of all strings over \( \{0, 1\} \) that have an equal number of 0’s and 1’s. The first tape contains the input, and is scanned from left to right. The second tape is used to keep track of the difference between the number of 0’s and 1’s in the part of the input seen so far.

\[
M = (\{q_0, q_1, q_2\}, q_0, q_2, \{Λ, 0, 1, \#\}, \{0, 1\}, Λ, δ)
\]

The machine starts in state \( q_0 \). If the input string is empty, it goes directly to accepting state \( q_2 \). If the input string is non-empty, the machine writes \( \# \) on the second tape and goes to state \( q_1 \).

- for each 0 on tape 1, the machine moves left on tape 2
- for each 1 on tape 1, the machine moves right on tape 2

If after the whole word is scanned on tape 1, the head of tape 2 is on symbol \( \# \), the machine accepts by going to accepting state \( q_2 \); otherwise it rejects.

\[
\delta(q_0, [Λ, Λ]) = (q_2, (Λ, R), (Λ, R))
\]
\[
\delta(q_0, [0, Λ]) = (q_1, (0, R), (\#, L))
\]
\[
\delta(q_0, [1, Λ]) = (q_1, (1, R), (\#, R))
\]
\[
\delta(q_1, [0, Λ]) = (q_1, (0, R), (Λ, L))
\]
\[
\delta(q_1, [1, Λ]) = (q_1, (1, R), (Λ, R))
\]
\[
\delta(q_1, [0, \#]) = (q_1, (0, R), (\#, L))
\]
\[
\delta(q_1, [1, \#]) = (q_1, (1, R), (\#, R))
\]
\[
\delta(q_1, [Λ, Λ]) = (q_2, (Λ, R), (\#, R))
\]

**Problem 9.** For a TM \( M \), let \( E(M) \) denote the encoding of \( M \). Consider the language

\[
L = \{ E(M) \mid M, \text{ when started on a blank tape eventually writes a 1 somewhere on the tape} \}
\]

(i) Show that \( L \) is recursively enumerable.
(ii) Show that $L$ is not recursive.

Let’s use notation $\langle M \rangle$ from the lectures for $E(M)$.

(i) We can construct a TM $M_L$ that recognizes $L$ as follows. $M_L$ first transforms its input $\langle M \rangle$ to $\langle\langle M \rangle, \epsilon\rangle$. $M_L$ simulates a universal TM $U$ except the following cases:

- as soon as $U$ writes a 1 on the tape, $M_L$ moves to an accepting state
- if $U$ halts in an accepting state, $M_L$ halts in a non-accepting state

(ii) $L_{\text{blankhalt}} = \{\langle M_{\text{blankhalt}} \rangle \mid M_{\text{blankhalt}}(\epsilon) \text{ halts} \}$ is a halting language. It is the language of encodings of TMs that halt on the blank tape. $L_{\text{blankhalt}}$ is not recursive (as was proven on lectures).

We show how to reduce $L_{\text{blankhalt}}$ to $L$. Assume there is a decider $D$ such that $D(\langle M \rangle)$ halts in an accepting state if and only if $\langle M \rangle \in L$ and otherwise halts in a non-accepting state. We show that using $D$ we can construct $D_{\text{blankhalt}}$ such that $D_{\text{blankhalt}}(\langle M_{\text{blankhalt}} \rangle)$ halts in an accepting state if and only if $\langle M_{\text{blankhalt}} \rangle \in L_{\text{blankhalt}}$ and otherwise halts in a non-accepting state.

We define a converter $R$ such that for all machines $M_{\text{blankhalt}}$ such that $\langle M_{\text{blankhalt}} \rangle \in L_{\text{blankhalt}}$, it returns machine $M$ such that $\langle M \rangle \in L$.

$R$ constructs $M$ from $M_{\text{blankhalt}}$ in the following way:

- All occurrences of symbol 1 in the transitions of $M_{\text{blankhalt}}$ (read and written) are replaced in $M$ with $\#$, where $\#$ is a new symbol not occurring in the tape alphabet of $M$.
- The final states of $M_{\text{blankhalt}}$ are not final in $M$. Instead, for each final state $q_f$ of $M_{\text{blankhalt}}$, for every tape symbol $x$, $R$ adds a transition $\delta(q_f, x) = (q_f, 1, R)$ to $M$.
- For each state $q$ and tape symbol $x$ for which $\delta(q, x)$ is undefined in $M_{\text{blankhalt}}$ (i.e. $M_{\text{blankhalt}}$ halts on $x$), $R$ adds a transition $\delta(q, x) = (q_f, 1, R)$ to $M$.

$M_{\text{blankhalt}}$ eventually halts when started on a blank tape if and only if $M = R(M_{\text{blankhalt}})$ eventually writes 1 somewhere on the tape.

Then we can take $D_{\text{blankhalt}} = D(\langle R(M_{\text{blankhalt}}) \rangle)$.

We know such a decider does not exist because $L_{\text{blankhalt}}$ is not recursive.