Introduction to the Theory of Computation 2022 — Midterm 1

Solutions

Problem 1 (20 pts). With the alphabet

$$\Sigma = \{0, 1\},$$

we consider the languages

$$L_1 = \{\boldsymbol{w} \mid \text{sum}(\boldsymbol{w}) = 1\}$$

and

$$L_2 = \{\boldsymbol{w} \mid \boldsymbol{w} \text{ ends with } 1\},$$

where we define

$$\text{sum}(\boldsymbol{w}) = w_1 + w_2 + \cdots + w_n \text{ if } \boldsymbol{w} = w_1 w_2 \ldots w_n.$$

(a) (5 pts) Give NFAs respectively for L_1 and L_2, each of which has ≤ 2 states. Then, follow the procedure in our slides "chap1_NFA4.pdf" from page 8 to page 11 (Theorem 1.47 in our textbook) to give the NFA that recognizes $L_1 \circ L_2$. Note that you need simplify the NFA such that the final NFA has ≤ 3 states.

(b) (5 pts) Convert your NFA in (a) to a DFA by the procedure in our slides "chap1_NFA3.pdf" from page 3 to page 9 (Theorem 1.39 in our textbook.) After that, show a figure without unused states.

(c) (10 pts) Prove that $L_1 \circ L_2$ is equivalent to the language

$$A = \{\boldsymbol{w} \mid \text{sum}(\boldsymbol{w}) \geq 2 \text{ and } \boldsymbol{w} \text{ ends with } 1\}$$

by showing that

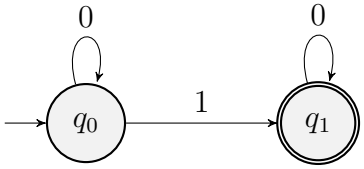
$$L_1 \circ L_2 \subseteq A$$

and

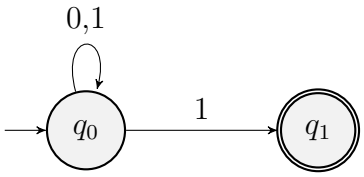
$$A \subseteq L_1 \circ L_2.$$

Solution.

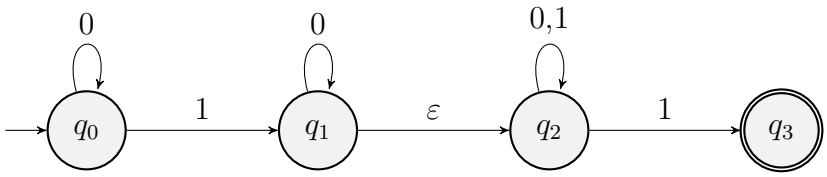
(a) Let us begin from the NFAs of L_1 and L_2 first. To recognize L_1, we construct the following NFA.



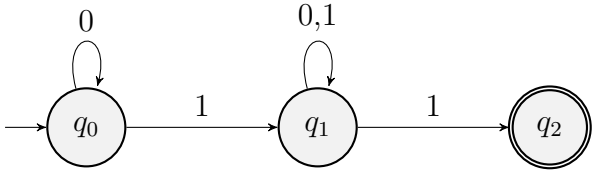
For L_2, we use



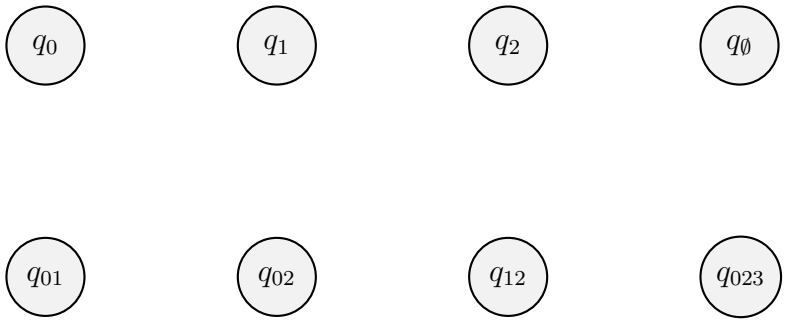
to recognize it. Hence, we can apply



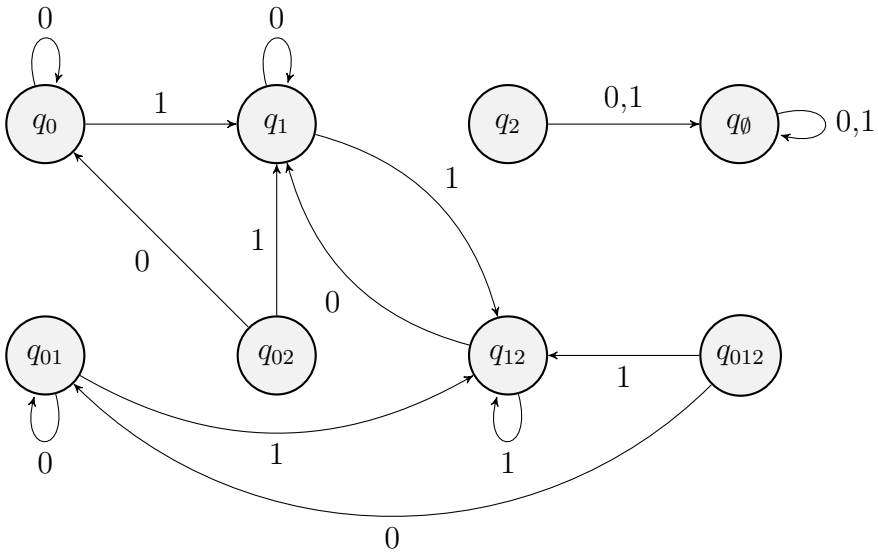
to recognize $L_1 \circ L_2$, and further simplify it to



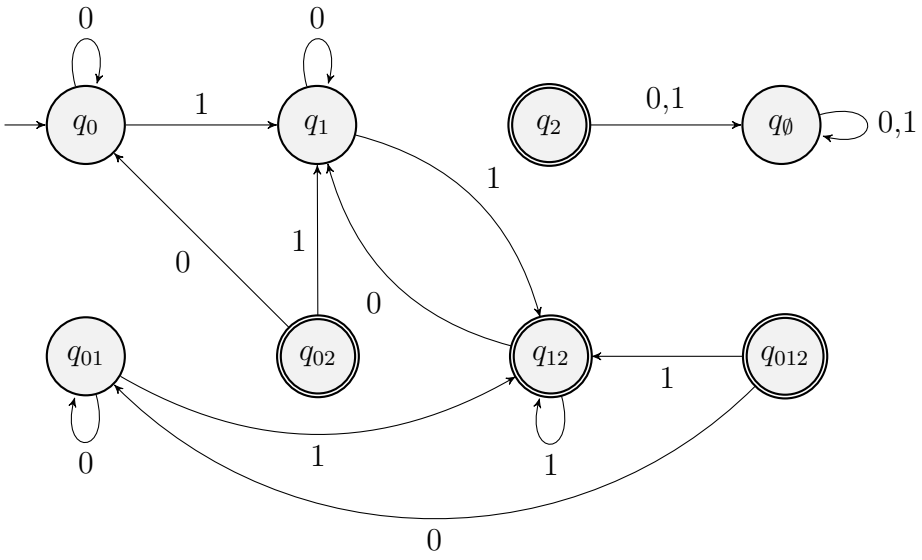
(b) Let us list all the elements of $P(Q)$ in the beginning.



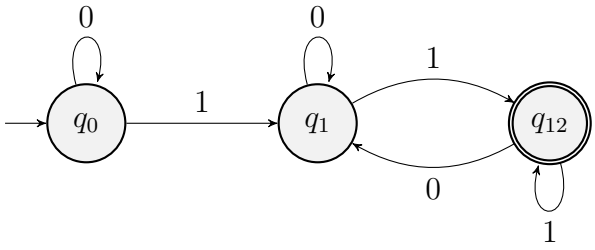
Then, we find out all possible paths.



Next, we decide the initial and accept states.



In the final, we remove unused states to simplify the diagram.



(c)

- $L_1 \circ L_2 \subseteq A$. Given the words $\boldsymbol{w}_1 \in L_1$ and $\boldsymbol{w}_2 \in L_2$, we have

$$\boldsymbol{w} = \boldsymbol{w}_1 \circ \boldsymbol{w}_2 \in L_1 \circ L_2.$$

Because $\boldsymbol{w}_2$ ends with "1", it implies that

$$\text{sum}(\boldsymbol{w}_2) \geq 1.$$

Thus,

$$\text{sum}(\boldsymbol{w}) = \text{sum}(\boldsymbol{w}_1) + \text{sum}(\boldsymbol{w}_2) \geq 2.$$

Furthermore, the end word of $\boldsymbol{w}$ is also equal to the end word of $\boldsymbol{w}_2$, which is "1", so

$$\boldsymbol{w} \in A.$$

- $A \subseteq L_1 \circ L_2$. Given a word $\boldsymbol{w} = w_1 \dots w_n \in A$, there exists an index $i \in \mathbb{N}$ such that

$$w_1 = \cdots = w_{i-1} = 0 \text{ and } w_i = 1. \tag{1}$$

Moreover,

$$i \neq n$$

because

$$\text{sum}(\boldsymbol{w}) = 1 \text{ if } i = n,$$

which implies

$$\boldsymbol{w} \notin A.$$

Hence, let us denote $\boldsymbol{w}_1 = w_1 \dots w_i$, and we can derive that

$$\boldsymbol{w}_1 \in L_1$$

by (1). Also, since $\boldsymbol{w} \in A$, $w_n = 1$. Then, it implies that

$$w_{i+1} \dots w_n \in L_2,$$

where we can take

$$\boldsymbol{w}_2 = w_{i+1} \dots w_n.$$

Clearly,

$$\boldsymbol{w} = \boldsymbol{w}_1 \circ \boldsymbol{w}_2 \in L_1 \circ L_2.$$

Therefore, we have derived that

$$L_1 \circ L_2 = A.$$

Note that we can draw a DFA to recognize L without any NFA, but usually it is difficult to find an equivalent language from the concatenation of two languages.

Common mistake: to prove

$$A \subseteq L_1 \circ L_2,$$

you must show that for any $\boldsymbol{w} \in A$, we can have

$$\boldsymbol{w} = \boldsymbol{w}_1 \circ \boldsymbol{w}_2$$

with $\boldsymbol{w}_1 \in L_1$ and $\boldsymbol{w}_2 \in L_2$.

Problem 2 (30 pts).

(a) (5 pts) Construct the NFA of the regular expression

$$1^*$$

using only one state. Give both the formal definition and its diagram. For this NFA, use

$$\Sigma_1 = \{1\}$$

as the alphabet in the definition.

(b) (5 pts) Notice that on page 10 of the slide "chap1_NFA4.pdf", we require two NFA to have the same alphabet to be concatenated. This does not sacrifice any generality, since any NFA with alphabet Σ' can be expanded to an equivalent NFA with a larger alphabet Σ where $\Sigma' \subset \Sigma$. Therefore, any two NFA can be expanded to share the same alphabet before regular operations such as union and concatenation.

(i) Given any NFA $(Q, \Sigma', \delta', q_0, F)$, show how to modify δ' to a new transition function δ so that $(Q, \Sigma, \delta, q_0, F)$ is an equivalent NFA.

(ii) Then, apply it to the NFA in subproblem (a) to give a new formal definition that uses the new alphabet

$$\Sigma = \{0, 1\}.$$

(c) (10 pts) Construct an NFA for the regular expression

$$R = (01^*)^*$$

by the following steps:

(i) Give the NFA diagram for regular expression

$$R_1 = 0$$

using at most two states.

(ii) Using the method from slide "chap1_NFA4.pdf", concatenate the NFA from subproblem (ci) with the NFA from subproblem (a) to form the NFA diagram for the regular expression

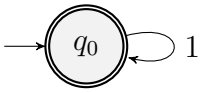
$$R_2 = 01^*.$$

(iii) Using the method from slide "chap1_NFA4.pdf", perform the star operation on the NFA from subproblem (cii) to give the diagram for the regular expression R.

(d) (10 pts) Convert the NFA diagram from subproblem (ciii) to a equivalent GNFA directly.[1] Then convert the GNFA back to a regular expression that is different (but equivalent) to the original expression R. You should show the initial GNFA and the GNFA after each removal of a state.

Solution.

(a) The regular expression 1^* can be recognized by this NFA:



It has the formal definition:

$$\begin{aligned} M =& (Q, \Sigma_1, \delta, q_0, F) \\ Q =& \{q_0\} \\ F =& \{q_0\} \\ \delta =& \end{aligned}$$

	1	ϵ
q_0	$\{q_0\}$	$\{\}$

(b)

(i) Given any NFA $(Q, \Sigma', \delta', q_0, F)$, we can enlarge the alphabet but does not allow any transition using the new symbols. This way, the new NFA would still recognize the same language. Formally, we define the new NFA as:

$$N = (Q, \Sigma, \delta, q_0, F)$$

$$\delta(q, s) = \begin{cases} \delta'(q, s) & \text{if } s \in \Sigma'_\epsilon \\ \{\} & \text{if } s \notin \Sigma'_\epsilon \end{cases}, \forall q \in Q, s \in \Sigma_\epsilon$$

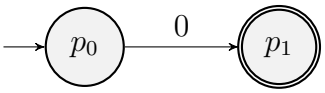
Common mistake: You need Σ'_ϵ instead of Σ'. Note that $\epsilon \notin \Sigma'$.

[1] The text book only showed how to convert a DFA to a GNFA, but let's apply the same procedure on an NFA.

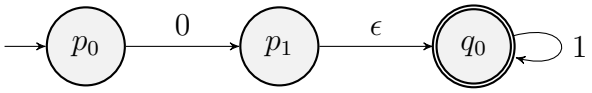
(ii) Applying this to the NFA from (a), we get:

$$
\begin{aligned}
M =& (Q, \Sigma, \delta, q_0, F) \\
Q =& \{q_0\} \\
F =& \{q_0\} \\
\delta =& \begin{array}{c|ccc} & 0 & 1 & \epsilon \\ \hline q_0 & \{\} & \{q_0\} & \{\} \end{array}
\end{aligned}
$$

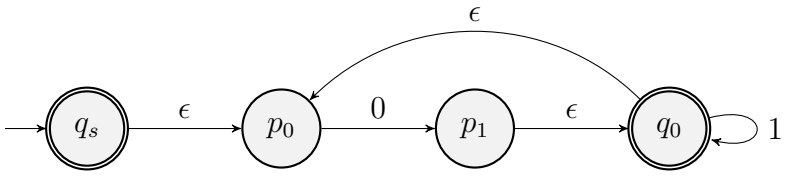
(c) (i) The NFA for 0 is



(ii) Concatenating the NFA from (ci) and (a), we get

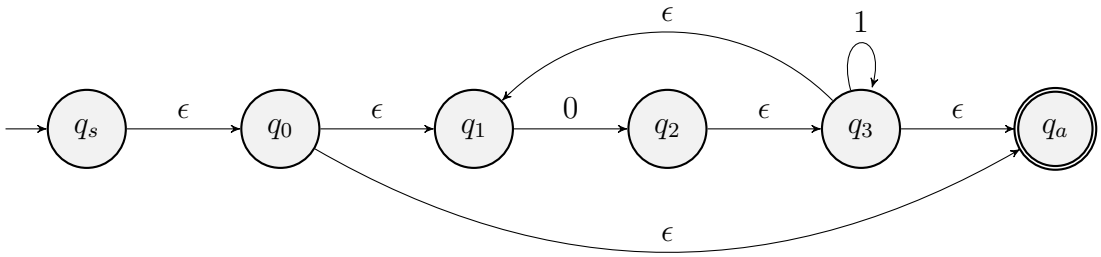


(iii) Applying the star operation on the NFA from (cii), we get

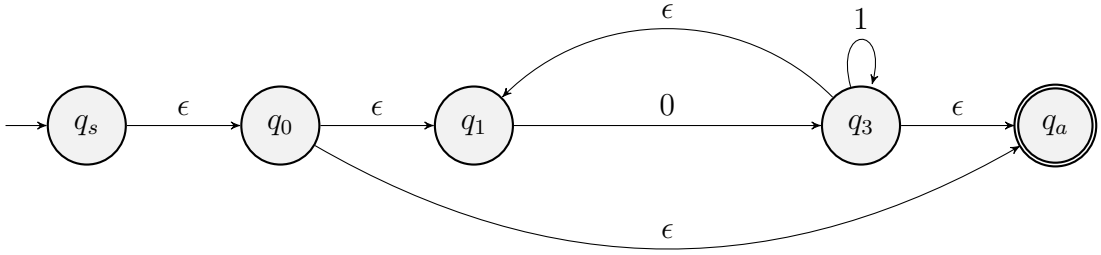


Common mistake: You need to have four states as we specifically asked you to follow the slides. If you only have three states, you need to explain why no extra strings are accepted.

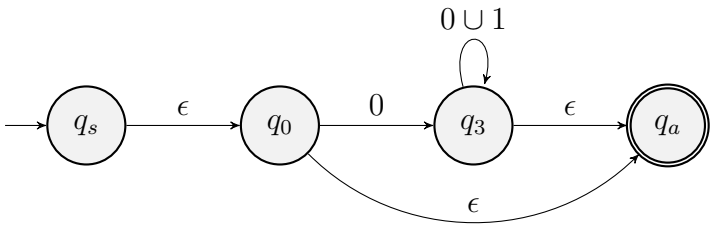
(d) In the text book, it only showed how to convert a DFA to a GNFA. However, NFA can also be converted to GNFA directly. To do this, we only have to add a new start and accept state:



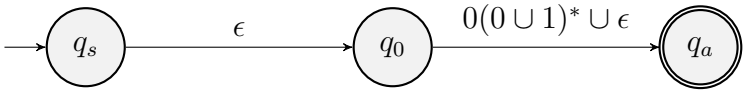
The states are renamed for convenience. Firstly, we can remove q_2 and get



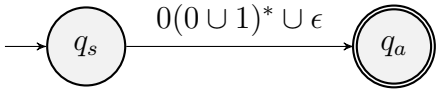
since $0\emptyset^*\epsilon \cup \emptyset = 0$. Then, we remove q_1 to get:



After that, remove q_3 and so we have:



Finally, remove q_0 so we get:



Therefore, the regular expression

$$0(0 \cup 1)^* \cup \epsilon$$

is equivalent to

$$(01^*)^*.$$

Problem 3 (20 pts). Let

$$\Sigma_2 = \left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

That is, Σ_2 contains all columns of 0s and 1s of height two. Each string $\boldsymbol{w} \in \Sigma_2^*$ is written as

$$\boldsymbol{w} = \begin{bmatrix}a_1\\b_1\end{bmatrix}\begin{bmatrix}a_2\\b_2\end{bmatrix}\ldots\begin{bmatrix}a_n\\b_n\end{bmatrix},$$

where $n \geq 0$, a_i and b_i are either 0 or 1 for each $1 \leq i \leq n$. Consider the language

$$L = \{\boldsymbol{w} \in \Sigma_2^* \mid b_1 b_2 \ldots b_n \text{ is the outcome of right shifting } a_1 a_2 \ldots a_n \text{ by one bit}\}.$$

For simplicity, we assume the leftmost bit after applying the shift operation is always 0. For example,

$$\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} \in L, \ \begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \in L, \text{ while } \begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \notin L.$$

Note that $\epsilon \in L$ since the outcome of right shifting an empty string is still an empty string.

(a) (10 pts) Construct a DFA for L in less than or equal to 3 states

(b) (10 pts) Prove that L can not be recognized by a DFA with less than 3 states. Our proof is by contradiction. Assume L can be recognized by a 2-state DFA. Then we must have the following situations (here q_0 is not necessary a start state):

(1)

or

(2)

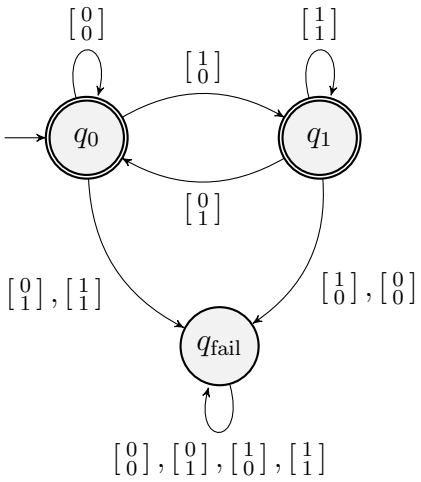
or

(3)

Explain which cases are not possible. From the remaining case(s), finish the proof.

Solution.

(a) L can be recognized by the following DFA:



(b) Case (1) is impossible, since we need a reject state to reject $\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}$. Case (3) is impossible, either, since we need at least an accept state to accept $\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}$. Therefore, case (2) is the only remaining situation.

For case (2), since ϵ is accepted, q_0 must be the start state. Then the DFA must be of the following form:

We then focus on the transition $\delta(q_0, \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right])$. If $\delta(q_0, \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]) = q_0$, then $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \notin L$ would be accepted. However, if $\delta(q_0, \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]) = q_1$, then $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \in L$ would be rejected. This is a contradiction.

Problem 4 (30 pts). For the language in each subproblem, the alphabet $\Sigma = \{0, 1\}$.

(a) Consider

$$A = \{\boldsymbol{w} \in \Sigma^* \mid \boldsymbol{w} \text{ has an equal number of occurrences of 010 and 101 as substrings}\}.$$

We try to prove that A is not regular by the pumping lemma. Assume for contradiction that A is regular with pumping length p. Consider

$$s = (010)^p 01 (101)^p \in A.$$

According to the pumping lemma, s can be written as xyz with $|y| > 0$, $xy^iz \in A$ for all $i \geq 0$ and $|xy| \leq p$.

(i) (10 pts) Can we finish the proof by showing that $xy^0z \notin A$ for all possibilities of x, y and z?

(ii) (5 pts) Can we finish the proof by showing that $xy^2z \notin A$ for all possibilities of x, y and z?

(b) Consider

$$B = \{\boldsymbol{w} \in \Sigma^* \mid \boldsymbol{w} \text{ has an equal number of occurrences of 011 and 110 as substrings}\}.$$

We would like to show that B is regular. For simplicity, we denote #011 and #110 the number of occurrences of 011 and 110 respectively.

(i) (10 pts) We want to show that $|\#011 - \#110| \leq 1$ holds for any $\boldsymbol{w}$. Let n be the length of $\boldsymbol{w}$. The statement is trivial for $n \leq 3$, since both #011 and #110 are no more than 1 for $n \leq 3$. For any $n > 3$, we can prove by induction. For any string, we define the following situations:

$$\begin{aligned}
0_s&: \text{ the last character is 0 and } \#011 = \#110\\
01_s&: \text{ the last 2 characters are 01 and } \#011 = \#110\\
11_s&: \text{ the last 2 characters are 11 and } \#011 = \#110\\
11_+&: \text{ the last 2 characters are 11 and } \#011 = \#110 + 1\\
0_-&: \text{ the last character is 0 and } \#011 = \#110 - 1\\
01_-&: \text{ the last 2 characters are 01 and } \#011 = \#110 - 1.
\end{aligned}$$

Please prove that

$$\begin{aligned}
&\text{if } \boldsymbol{w}_{1:n-1} = w_1 \dots w_{n-1} \text{ is in one of these situations,}\\
&\text{then } \boldsymbol{w} = w_1 \dots w_n \text{ is also in one of these situations.}
\end{aligned}$$

(Then by induction, each $\boldsymbol{w}$ is in one of these situations. You don't need to consider the initial condition in this problem.)

(ii) (5 pts) Based on results in (bi), design a DFA with no more than 8 states for the language.

Solution.

(a) (i) We can obtain a contradiction by showing that $xy^0z \notin A$ for all possible decomposition of $s = xyz$. We splits all possible y's into the following cases.

Case 1: y contains at least one "1".
Since there are only p 1's in the substring $(010)^p$ of s, the number of 010 must decrease if we remove y. Therefore, $xy^0z \notin A$.

Case 2: $y = 0$, $xy^0z = (010)^k(10 \cup 01)(010)^{p-k}01(101)^p$.
For $k = 0$, $\#010 = p - 1$ and $\#101 = p$.
For $k > 0$ and $|xy| \leq p$, $\#010 = p$ and $\#101 = p + 1$.

Case 3: $y = 00$, then $xy^0z = (010)^k(0110)(010)^{p-k-2}01(101)^p$:
For $k \geq 0$ and $|xy| \leq p$, $\#010 = p - 2$ and $\#101 = p$.

Note: you must check all possible decompositions. If you consider a particular x and y and think your proof is right, then **you still don't quite understand the pumping lemma.**

(ii) We cannot obtain a contradiction by showing that $xy^2z \notin A$ for all possible decomposition of $s = xyz$. This is because for $x = \epsilon$, $y = 0$ and z being the remaining part, $xy^2z = 0(010)^p01(101)^p \in A$.

Comment: Originally, what we intend to prove is that

$$\forall p, \{\exists s \in A, |s| \geq p, [\forall x, y, z((s = xyz \text{ and } |y| > 0 \text{ and } |xy| \leq p) \rightarrow xy^2z \notin A)]\}.$$

In the sample solution, when $x = \epsilon$ and $y = 0$, the above holds under any positive integer p. However, it is sufficient that the chosen x and y are valid under a specific p. For example, when $p = 3$, we can see for

$$x = 01 \text{ and } y = 0,$$

$|xy| \leq p$, $|y| > 0$ and $xy^2z \in A$.

(b) (i) We prove the statement by enumerating all transitions among situations after appending 0 or 1 to $w_1 \dots w_{n-1}$.

w_n \ $w_1 \dots w_{n-1}$	0_s	01_s	11_s	11_+	0_-	01_-
0	0_s	0_s	0_-	0_s	0_-	0_-
1	01_s	11_+	11_s	11_+	01_-	11_s

(ii) The figure is like the following. Note that we need a temporary q' state to handle strings with "1" as the first character.