HW 3 Solution

CS/ECE 374 B: Algorithms & Models of Computation, Spring 2020

Submitted by:

- **≪Michael Jiang≫**: **≪minhaoj2≫**
- **≪Yumai Sun≫**: **≪yumais2≫**
- **≪Shuchen Zhang≫**: **≪szhan114≫**

7 Problem 7:

Solution 7.A:

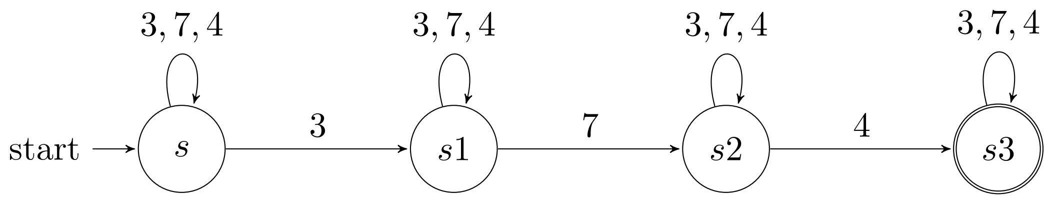

Figure 1: NFA

Since the $\sum = \{3, 7, 4\}$, for any transition with $\sum^*$ we can let any input in this set loop on the same node until we accept the next valid input. In this case, this NFA would accept any string in the context.

Solution 7.B:

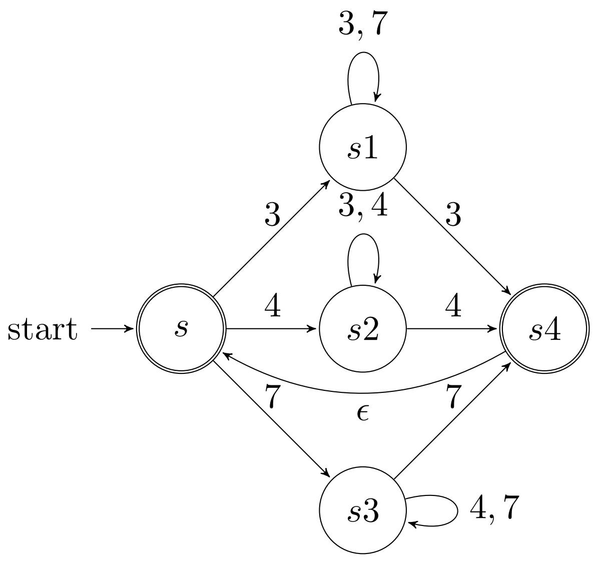

Figure 2: NFA

With Thompson's Algorithm, we can divide this regular expression into three cases, starting with 3,7,4 respectively. After that, we can simply follow the rule of concatenation of string and Kleene* to construct the NFA.

Solution 7.C:

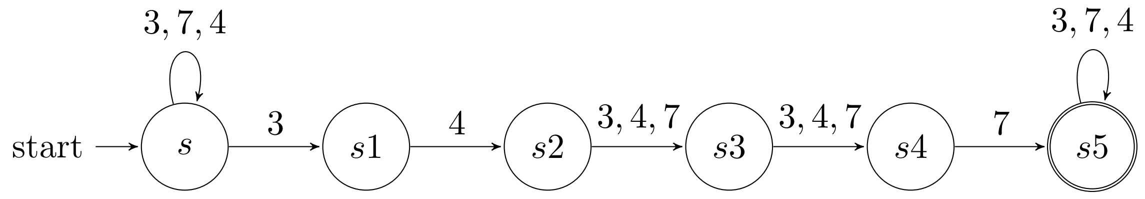

Figure 3: NFA

With the same reason as in 7.A, we can add self loop on the node for $\sum^*$ and then for $(3+7+4)^2$ transition, we can directly combine them on one edge. Therefore, this NFA would suffice.

Solution 7.D:

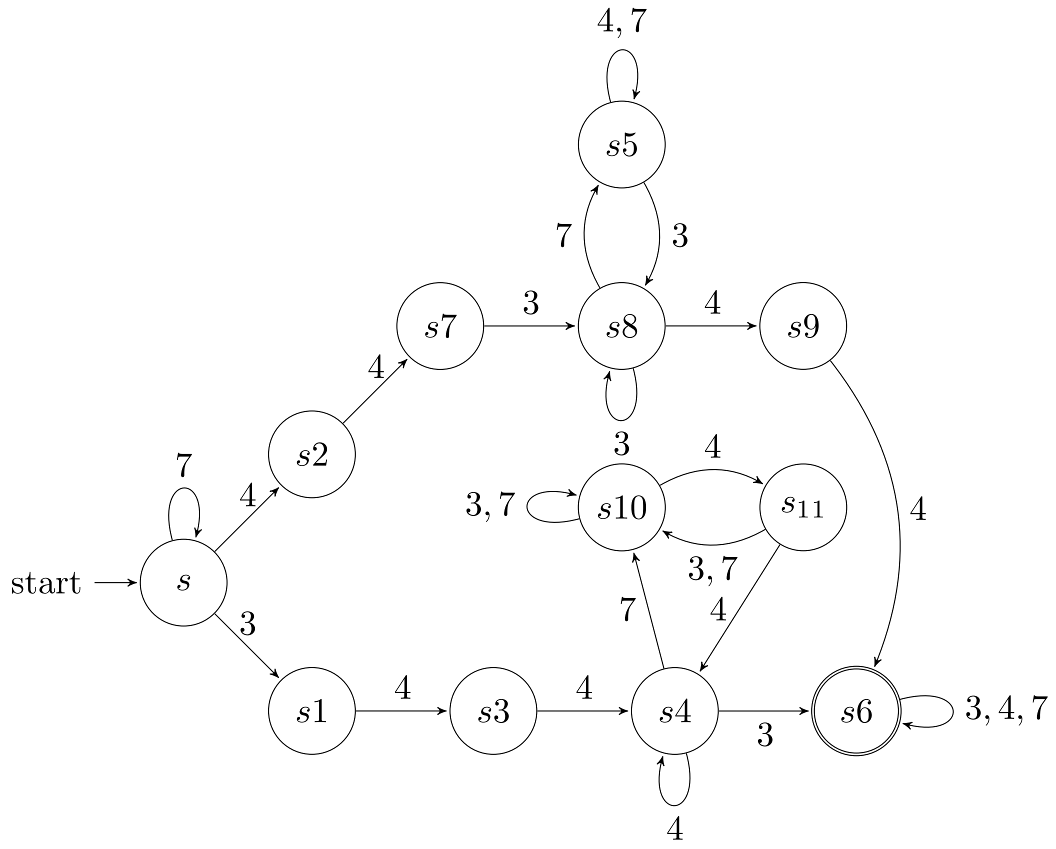

Figure 4: NFA

There are two general situations. One is that 443 appears before 344, and the other one is that 344 appears before 443. Since the two substrings have numbers that are the same, we could consider the cases when they overlap, just attach, and separated by some other states. Thus by adding states on top of one states could include all those cases, and then we union the two general cases together to get the NFA.

Solution 7.E:

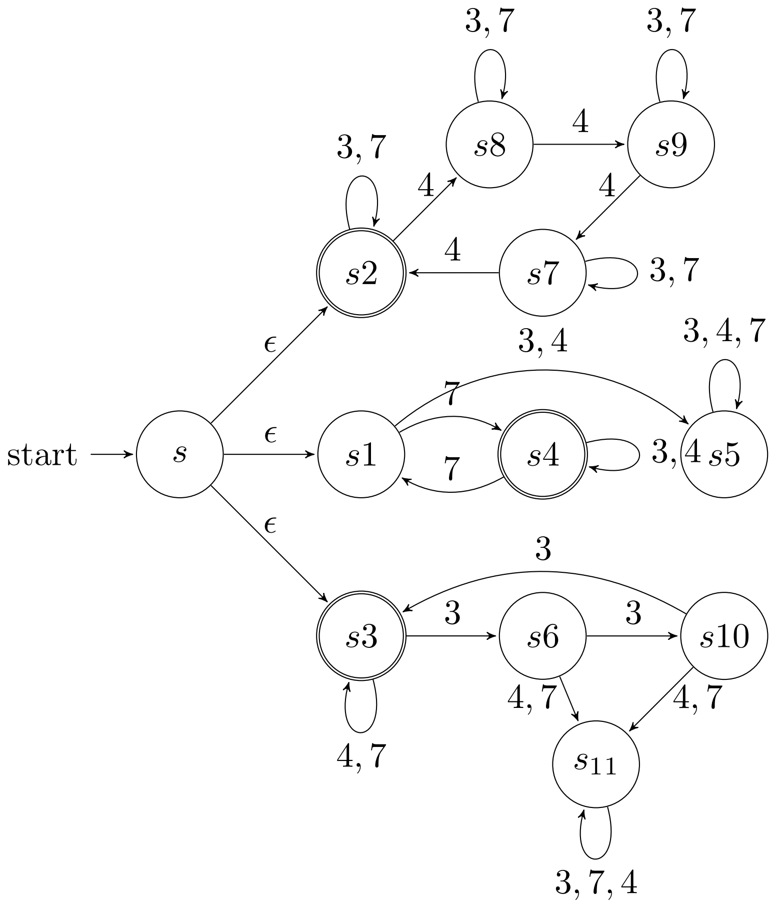

Figure 5: NFA

For this NFA, in order to have at least one condition be satisfied, we have constructed a start state that could free transfer to any of the three situations, so that we could check whether the input string satisfies one of the conditions. In these three conditions, we can easily construct a DFA corresponding to each condition so that this NFA would be able to describe the regular expression.

8 Problem 8:

Solution 8.A:

$$
\begin{aligned}
&Q' = Q\\
&\Sigma' = \{0, \epsilon\}\\
&s' = s\\
&A' = A\\
&\delta'(q,a) = \begin{cases} \delta(q,0), & \text{if } a = 0.\\ \delta(q,1), & \text{if } a = \epsilon. \end{cases}
\end{aligned}
$$

Since we want to remove all 1's in the string, we can simply convert all 1-transition in original DFA to $\epsilon-$ transition so that our new NFA can take the 1-transition in DFA as a free transition, so the machine would only take removed 1s strings .

Solution 8.B:

$$
\begin{aligned}
&Q' = Q\\
&\Sigma' = \Sigma\\
&s' = s\\
&A' = A\\
&\delta'(q,a) = \begin{cases} \delta(q,a), & \text{if } a = 0.\\ q, & \text{if } a = 1. \end{cases}\\
&\delta'(q,\epsilon) = q
\end{aligned}
$$

In order to check if the input string without all the 1s is accepted by L, we will stay at the same state when we encounter a 1 in the input string, and transit in the DFA when we encountered a 0, so that we could check if we end up in the accept state with all the 1s removed from the input string.

Solution 8.C:

$$\begin{aligned}
&Q' = Q \times \{0, 1, 2, 3..., k\}\\
&\Sigma' = \Sigma\\
&s' = (s, c = 0)\\
&A' = \{(p, c) | p \in A \text{ and } c = k\}\\
&\delta'((q, c), a) = \begin{cases} \text{if } a = 1 & \\ \{(\delta(q, a), c), (q, c + 1)\} & c < k \\ (\delta(q, a), k) & c = k \\ \text{if } a = 0 & \\ (\delta(q, 0), c) & \end{cases}
\end{aligned}$$

Because of the non deterministic characters of NFA, we construct the state of the NFA as (q, c), where the q is the states from the L DFA machine, and c keeps track of how many 1s we have read in. When we encounter a 1, we could treat the 1 as either the 1 that's in the original string, or the 1 that we inserted. There will be many possibilities of states that we go to, but as along as at least one of the path that ends up in a state that satisfy the condition (p, c), where p is in the accepted states in the DFA, and $c = k$, the string will satisfy the condition, which is that we have inserted k number of 1s into the string.

Solution 8.D:

$$\begin{aligned}
&Q' = Q \cup \{s'\}\\
&\Sigma' = \Sigma\\
&s' = s' \text{ A new state.}\\
&A' = \{p \in Q | \text{there exists strings } x, y \in \Sigma^* \text{ such that, } \delta^*(s, x) = p, \text{ and } \delta^*(p, y) \in A.\}\\
&\delta'(q, a) = \delta(q, a)\\
&\delta'(s', \epsilon) \in Q\\
&\delta'(s', a) = \emptyset
\end{aligned}$$

We added a new start state s' that can free transit to any state in the NFA, so that the input string could be starting from any where in our NFA. Then we make the accepted states to be the stats that 1. $\delta^*(s, x) = q \in Q$ where $x \in \Sigma^*$, i.e. the NFA will accept prefixes in DFA, and 2. $\delta^*(q, y) \in A$ where $y \in \Sigma^*$, i.e. the NFA will accept suffixes in DFA. The ϵ transition in the beginning, made sure that we could accept any type of substrings of DFA.

9 Problem 9:

Solution 9.A:

#1

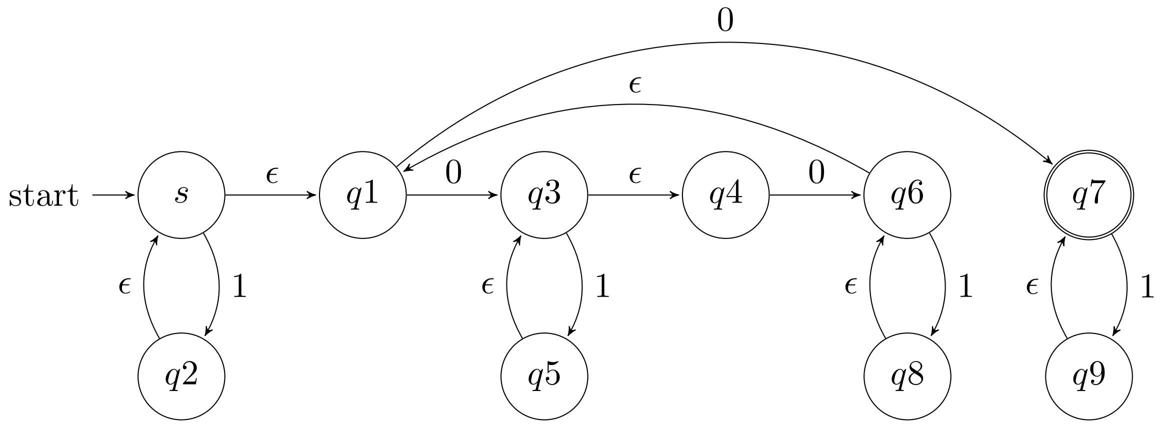

Figure 6: NFA

#2

q'	ϵ reach	A?	$\delta(q,0)$	$\delta(q,1)$
s	s, q_1	No	q_3q_7	q_2
q_3q_7	$q_3q_4q_{10}$	Yes	q_6	$q_5q_7q_9$
q_2	sq_1q_2	No	q_3q_7	q_2
q_6	q_1q_6	No	q_3q_7	q_8
$q_5q_7q_9$	$q_3q_4q_5q_7q_9$	Yes	q_6	$q_5q_7q_9$
q_8	$q_1q_6q_8$	No	q_3q_7	q_8

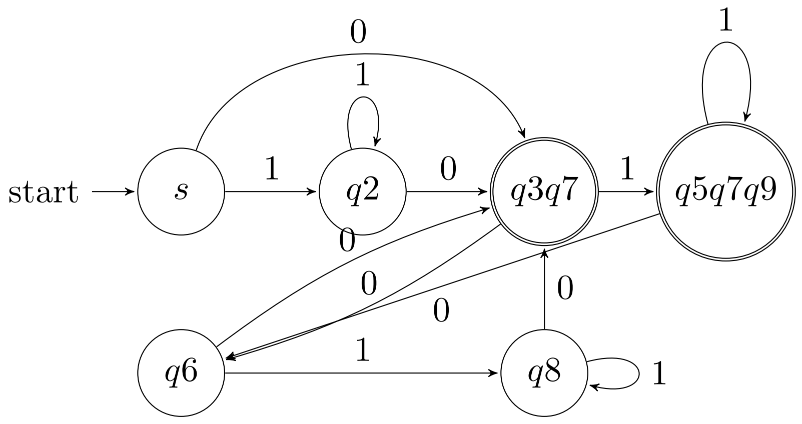

Figure 7: NFA

#**3**

$L(N) = \{\omega \mid \omega \text{ with odd number of 0s}\}$

#**4**

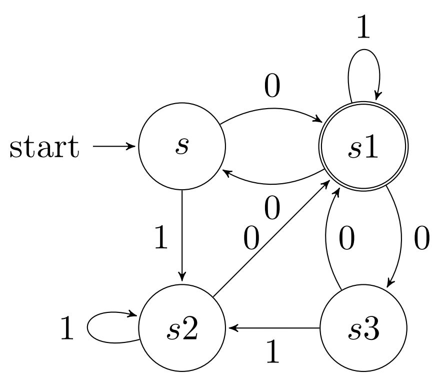

Figure 8: DFA

Solution 9.B:

#1

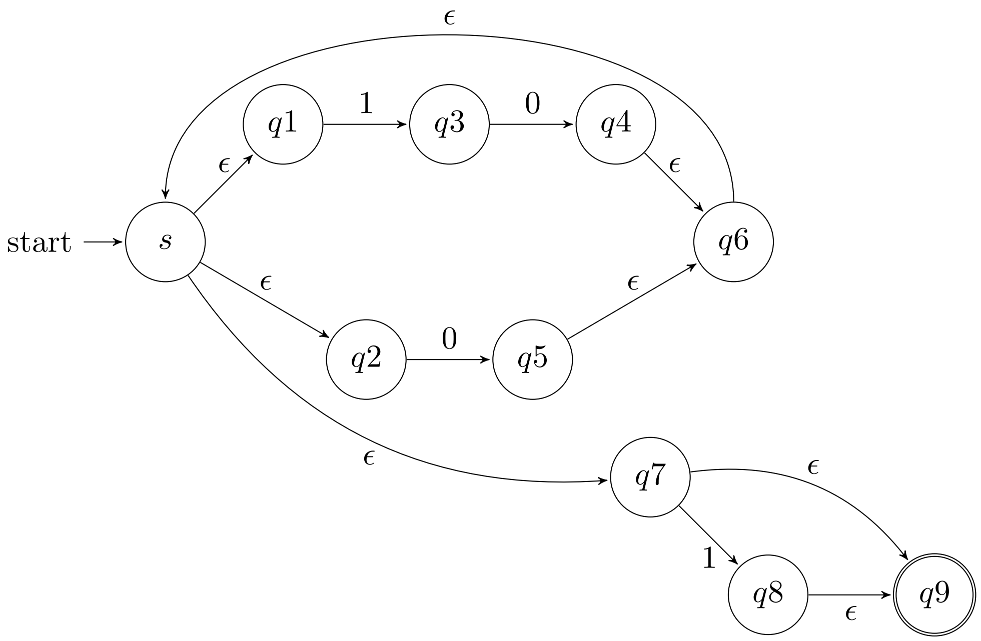

Figure 9: NFA

#2

q'	ϵ reach	A?	$\delta(q,0)$	$\delta(q,1)$
s	$sq_1q_2q_7q_9$	Yes	q_5	q_3q_8
q_5	$sq_1q_2q_5q_6q_6q_8q_9$	Yes	q_5	q_3q_8
q_3q_8	$q_3q_8q_9$	Yes	q_8	$\emptyset$
q_8	$sq_1q_4q_6q_2q_7q_9$	Yes	q_5	q_3q_8
$\emptyset$	$\times$	No	$\emptyset$	$\emptyset$

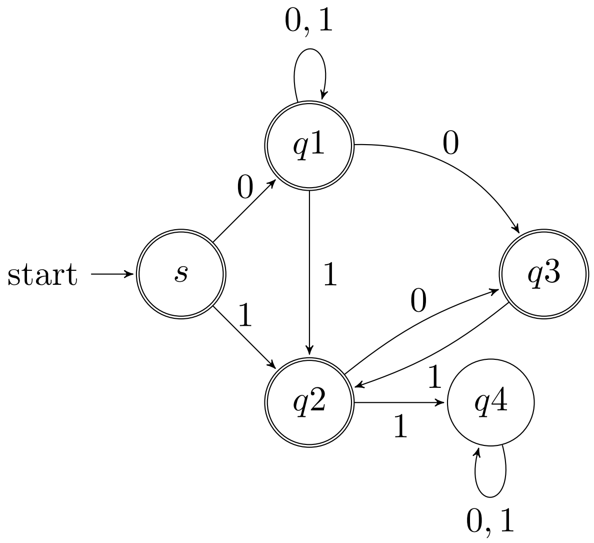

Figure 10: NFA

#**3**

$L(N) = \{\omega \mid \omega$ does not have two consecutive 1s$\}$

#**4**

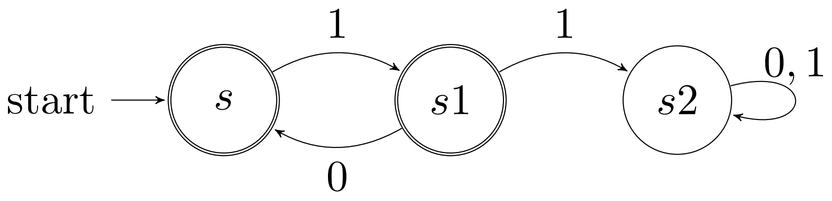

Figure 11: DFA