

# Homework 1 Solutions

## Formal Languages and Automata (CS322)

Last update: October 14, 2025

1. Fix an alphabet  $\Sigma$  (i.e., a nonempty finite set of symbols). For a string  $s \in \Sigma^*$ , we denote its length by  $|s|$ . For two strings  $s, t \in \Sigma^*$ , we write  $s \sqsubseteq t$  to mean that  $s$  is a substring of  $t$  (i.e., there exist strings  $x, y \in \Sigma^*$  such that  $t = xsy$ ). The *substring problem* has the following input and output:

- Input: two strings  $s, t \in \Sigma^*$ .
- Output: YES if  $s \sqsubseteq t$ , NO otherwise.

(Remark: In L<sup>A</sup>T<sub>E</sub>X,  $\sqsubseteq$  is `\sqsubseteq` or `\subseteq`.)

- (a) Let  $s \in \Sigma^*$  be an arbitrary string. Define a language  $L_s = \{t \in \Sigma^* : s \sqsubseteq t\}$ . Construct a deterministic finite automaton (DFA) with at most  $|s| + 1$  states that recognizes  $L_s$ . Justify your answer. (Remark: You cannot express the answer using a transition diagram since the answer depends on an arbitrary string  $s$ . You must describe a transition function (as well as other entries of a DFA) in a well-defined way.)
- (b) Present an algorithm for the substring problem in time  $O(|s| + |t|)$ . Justify your answer (i.e., provide a correctness proof and running-time analysis).

*Solution.* (Student's solution by Jaemin Lim)

### (a) Construction

For  $x, y \in \Sigma^*$ , let's define that  $x$  is a suffix of  $y$  if  $y = px$  for some  $p \in \Sigma^*$  and write  $x \leq y$ .

Let  $n = |s|$  and  $s = s_1 s_2 \cdots s_n$ . For any  $i = 0, 1, \dots, n$  and  $c \in \Sigma$ , define

$$J(i, c) = \{j = 0, 1, \dots, n : s_1 s_2 \cdots s_j \leq s_1 s_2 \cdots s_i c\}$$

If  $j = 0$ , then  $s_1 s_2 \cdots s_j = \epsilon$ .  $J(i, c)$  is never empty because  $\epsilon$  is a suffix of any string as  $z = z\epsilon$  for any  $z \in \Sigma^*$ , so  $0 \in J(i, c)$ .

Let  $q_0, q_1, \dots, q_n$  be  $n + 1$  states. Define  $Q = \{q_0, q_1, \dots, q_n\}$  and  $\delta: Q \times \Sigma \rightarrow Q$  as follows:

$$\delta(q_i, c) = \begin{cases} q_{\max J(i, c)} & \text{if } i < n \\ q_n & \text{if } i = n \end{cases}$$

Then the DFA given by  $M = (Q, \Sigma, \delta, q_0, \{q_n\})$  recognizes  $L_s$ .

### Justification

We want to show that, for  $t \in \Sigma^*$ ,

$$t \in L_s \iff M \text{ accepts } t$$

We prove each direction separately.

( $\implies$ ) Suppose  $t \in L_s$ , then there exists  $x, y \in \Sigma^*$  such that  $t = xsy$ . For each  $i = 0, 1, \dots, n$ , consider the statement

$$P(i): \text{ If } \hat{\delta}(q_0, xs_1 \cdots s_i) = q_k, \text{ then } (k \geq i, \\ \text{and if } k < n, \text{ then } (s_j = s_{j+k-i} \text{ for all } j = 1, 2, \dots, i)).$$

We show that  $P(i)$  holds for all  $i = 0, 1, \dots, n$ . The proof is by induction on  $i$ .

$P(0)$  is clearly true because if  $q_k \in Q$ ,  $k \geq 0$ , and there are no possible values for  $j$ , leaving the second part of  $P(0)$  vacuously true.

Assume that  $P(i)$  holds for some  $i = 0, 1, \dots, n-1$ . Let  $\hat{\delta}(q_0, xs_1 \dots s_i) = q_k$ ,  $k \geq i$ . Then we have  $\hat{\delta}(q_0, xs_1 \dots s_i s_{i+1}) = \delta(\hat{\delta}(q_0, xs_1 \dots s_i), s_{i+1}) = \delta(q_k, s_{i+1})$ . If  $k = n$ , then  $\delta(q_k, s_{i+1}) = q_n$  and  $n \geq i+1$ , so we are done. Therefore now assume that  $i \leq k < n$ .

By the second part of  $P(i)$ , we have

$$(s_1 \dots s_k) s_{i+1} = (s_1 \dots s_{k-i})(s_{k-i+1} \dots s_k) s_{i+1} = (s_1 \dots s_{k-i})(s_1 \dots s_i) s_{i+1}$$

which gives  $s_1 \dots s_{i+1} \leq (s_1 \dots s_k) s_{i+1}$  and therefore  $i+1 \in J(k, s_{i+1})$ . Therefore,  $k' := \max J \geq i+1$ , so  $\delta(q_k, s_{i+1}) = q_{k'}$  with  $k' \geq i+1$ , proving the first part of  $P(i+1)$ .

Since  $k' \in J(k, s_{i+1})$ , we have  $s_1 \dots s_{k'} \leq (s_1 \dots s_k) s_{i+1}$ . Noting that  $k-i+1 \geq 1$  and  $k'-i \geq 1$  and therefore  $s_{k-i+1}$  and  $s_{k'-i}$  are well-defined, we have

$$\begin{cases} s_{k'-i-1+j} = s_{k-i+j} = s_j & \text{for } j = 1, 2, \dots, i \\ s_{k'} = s_{i+1} \end{cases}$$

This means  $s_j = s_{j+k'-(i+1)}$  holds for all  $j = 1, 2, \dots, i+1$ , proving the second part of  $J(i+1)$  whenever  $k' < n$ .

Therefore by induction  $P(i)$  holds for all  $i = 0, 1, \dots, n$ , and in particular,  $P(n)$  holds. Define  $\hat{\delta}(q_0, xs) = q_k$ , then  $k \geq n$ , but if  $q_k \in Q$ , we must have  $k \leq n$ . Therefore,  $k = n$ .

Let  $y = y_1 y_2 \dots y_{|y|}$ . Then  $\hat{\delta}(q_0, xsy_1 \dots y_i) = q_n$  for all  $i = 0, 1, \dots, |y|$ . To prove this, we use induction on  $i$ . If  $i = 0$ , this is clear because  $xsy_1 \dots y_0 = xs$  by definition. Suppose  $\hat{\delta}(q_0, xsy_1 \dots y_i) = q_n$  for some  $i = 0, 1, \dots, |y|-1$ . Then  $\hat{\delta}(q_0, xsy_1 \dots y_i y_{i+1}) = \delta(\hat{\delta}(q_0, xsy_1 \dots y_i), y_{i+1}) = \delta(q_n, y_{i+1}) = q_n$ .

The above statement must, in particular, hold for  $i = |y|$ , giving  $\hat{\delta}(q_0, xsy) = q_n \in \{q_n\}$ , which means  $t = xsy$  is accepted by  $M$ .

( $\Leftarrow$ ) Suppose  $M$  accepts  $t$ . If  $s = \epsilon$ , then any string  $t \in \Sigma^*$  satisfies  $t \in L_s$  because  $t = \epsilon t$ , so there is nothing to prove. Therefore now let's assume  $n \geq 1$ .

Let  $t = t_1 t_2 \dots t_m$ , then  $\hat{\delta}(q_0, t_1 \dots t_m) = q_n$ , so the set defined as

$$I = \{i = 0, 1, \dots, m : \hat{\delta}(q_0, t_1 \dots t_i) = q_n\}$$

is nonempty. Therefore let  $k = \min I$ .

$t_1 \dots t_0 = \epsilon$  and  $\hat{\delta}(q_0, \epsilon) = q_0 \neq q_n$ , so  $k > 0$ . By definition of  $k$ ,  $q_r := \hat{\delta}(q_0, t_1 \dots t_{k-1}) \neq q_n$ . So we have  $\hat{\delta}(q_0, t_1 \dots t_k) = \delta(\hat{\delta}(q_0, t_1 \dots t_{k-1}), t_k) = \delta(q_r, t_k) = q_n$  and  $r < n$ , then by definition  $n = \max J(r, t_k)$ .

Therefore  $n \in J(r, t_k)$ , which means  $s_1 \dots s_n \leq s_1 \dots s_r t_k$ . This implies  $s_n = t_k$  and  $|s_1 \dots s_n| \leq |s_1 \dots s_r t_k| \implies n \leq r+1 \implies r \geq n-1 \implies r = n-1$  because  $r < n$ .

Consider the statement for  $i = 0, 1, \dots, k-1$ :

$$T(i): \text{ If } \hat{\delta}(q_0, t_1 \dots t_i) = q_v, \text{ then } s_1 \dots s_v \leq t_1 \dots t_i.$$

To show that  $T(i)$  holds for all  $i = 0, 1, \dots, k-1$ , we use induction on  $i$ .

$T(0)$  is clearly true because  $\hat{\delta}(q_0, t_1 \dots t_0) = \hat{\delta}(q_0, \epsilon) = q_0$  and  $s_1 \dots s_0 = \epsilon \leq \epsilon = t_1 \dots t_0$ .

Assume  $T(i)$  holds for some  $i = 0, 1, \dots, k-2$ . Let  $q_u = \hat{\delta}(q_0, t_1 \dots t_i)$  and  $q_v = \hat{\delta}(q_0, t_1 \dots t_{i+1})$ . If  $v = 0$ , then  $s_1 \dots s_v = \epsilon \leq t_1 \dots t_{i+1}$ , so  $T(i+1)$  is clearly true. Now assume  $v > 0$ . Since  $i+1 < k$ ,  $u < n$  and  $v < n$ . Now  $\hat{\delta}(q_0, t_1 \dots t_i t_{i+1}) = \delta(\hat{\delta}(q_0, t_1 \dots t_i), t_{i+1}) = \delta(q_u, t_{i+1}) = q_v$ . Therefore by definition  $v = \max J(u, t_{i+1})$ , meaning  $s_1 \dots s_v \leq s_1 \dots s_u t_{i+1}$ . Therefore,  $s_v = t_{i+1}$  and  $s_1 \dots s_{v-1} \leq s_1 \dots s_u \leq t_1 \dots t_i$ , where the second suffix relation comes from  $T(i)$ . Combining the results gives  $s_1 \dots s_{v-1} s_v \leq t_1 \dots t_i s_v = t_1 \dots t_i t_{i+1}$ , proving  $T(i+1)$ .

Therefore by induction  $T(i)$  holds for all  $i = 0, 1, \dots, k-1$ , and in particular,  $T(k-1)$  holds. Using the above result of  $r = n-1$  gives  $s_1 \dots s_{n-1} \leq t_1 \dots t_{k-1}$ , and combining it with  $s_n = t_k$

gives  $s \leq t_1 \cdots t_k$ . Therefore let  $t_1 \cdots t_k = xs$  for some string  $x \in \Sigma^*$  and let  $y = t_{k+1} \cdots t_m$ , then  $t = xsy$ , proving that  $t \in L_s$ .  $\square$

(Remark from TA: There were several students who constructed incorrect DFAs. Some failed to accept **ab** when  $s = \mathbf{a}$ , some failed to accept **aab** when  $s = \mathbf{ab}$ , and some failed to accept **aaab** when  $s = \mathbf{aab}$ . These examples may help you check yourself whether your answer is incorrect.)

(b) **Pseudocode**

```

n := |s|
m := |t|
prepare a list  $d_1, d_2, \dots, d_n$ 
j := 0
for (i := 1 to n)
    while (j > 0 and  $s_i \neq s_{j+1}$ )
        j ← d_j
        if (i > 1 and  $s_i = s_{j+1}$ ) j ← j + 1
        d_i ← j
j ← 0
for (i := 1 to m)
    if (j < n)
        while (j > 0 and  $t_i \neq s_{j+1}$ )
            j ← d_j
            if ( $t_i = s_{j+1}$ ) j ← j + 1
if (j = n) output Yes
else output No

```

**Justification: correctness - loop 1**

For  $i = 1, 2, \dots, n$ , define the set

$$S(i) = \{j = 0, 1, \dots, i-1 : s_1 \cdots s_j \leq s_1 \cdots s_i\}$$

The first **for** loop assigns the values of  $d_1, d_2, \dots, d_n$ . We claim that  $d_i = \max S(i)$ .

In a single iteration of the first **for** loop,  $j$  can increase by at most 1, while  $i$  always increases by 1. Since the starting value of  $j$  is 0 and  $i$  is 1,  $j \leq i$  always holds. This means in the iteration we never use the value of any  $d_j$  that is not already calculated.

This allows to use strong induction on  $i$  to prove the above claim. For  $i = 1$ ,  $j$  is initially 0 so the inner **while** loop doesn't run, which means  $d_1$  is assigned as 0 as expected.

Suppose  $d_k$  is correctly calculated for all  $k < i$ , where  $i$  is some value in  $2, 3, \dots, n$ . At the start of the **for** loop iteration with the current value  $i$ , the value of  $j$  directly comes from the end of previous iteration, which means  $j = d_{i-1}$ .

We claim two loop invariants for the inner **while** loop:  $j + 1 \geq \max S(i)$  and  $s_1 \cdots s_j \leq s_1 \cdots s_{i-1}$ . We induct on the loop count to prove these invariants. In the initial case, if  $d \in S(i)$ , then  $s_1 \cdots s_d \leq s_1 \cdots s_i$ . If  $d = 0$  then clearly  $d \leq j + 1$  because  $j$  is never negative, and if  $d > 0$  then  $s_1 \cdots s_{d-1} \leq s_1 \cdots s_{i-1} \implies d - 1 \in S(i - 1) \implies d - 1 \leq \max S(i - 1) = d_{i-1} = j$ , so  $d \leq j + 1$ . This holds for all  $d \in S(i)$ , so  $\max S(i) \leq j + 1$ . Also,  $j \in S(i - 1)$  directly implies  $s_1 \cdots s_j \leq s_1 \cdots s_{i-1}$ .

Suppose that the two invariant holds at the beginning of a **while** iteration. We have  $\max S(i) \leq j + 1$ , but since the loop did not terminate we also have  $s_i \neq s_{j+1}$ . This means  $s_1 \cdots s_j \leq s_1 \cdots s_i$  cannot hold and therefore  $j + 1 \notin S(i)$ , so  $\max S(i) < j + 1$ . Suppose  $k \in S(i)$ . Then  $s_1 \cdots s_k \leq s_1 \cdots s_i$ , so  $s_1 \cdots s_k \leq s_1 \cdots s_i$ , but we also have  $s_1 \cdots s_j \leq s_1 \cdots s_{i-1}$  and  $k \leq j$  because  $k \leq \max S(i) < j + 1$ . Therefore  $s_1 \cdots s_k \leq (s_1 \cdots s_j)s_i$ , meaning either  $k = 0$  or  $k - 1 \in S(j)$ , and in both cases  $k \leq d_j + 1$ . Therefore updating  $j \leftarrow d_j$  doesn't break the first invariant. The second invariant also keeps being true because  $d_j \in S(j) \implies s_1 \cdots s_{d_j} \leq s_1 \cdots s_j \leq s_1 \cdots s_{i-1}$ , so the update doesn't break the second invariant.

Eventually the **while** loop must terminate because  $j$  is strictly decreasing at each iteration and making  $j = 0$  breaks the loop. If it terminates because  $s_i = s_{j+1}$ , then by the loop invariants

$j + 1 \geq \max S(i)$  and  $s_1 \cdots s_j \leq s_1 \cdots s_{i-1} \implies s_1 \cdots s_j s_{j+1} \leq s_1 \cdots s_{i-1} s_i$ , where we are allowed to put one additional character to each string because they are equal, so  $j + 1 \in S(i) \implies j + 1 \leq \max S(i)$ . Therefore,  $\max S(i) = j + 1$ . In this case the increment of  $j$  inside the **if** conditional will run, so  $d_i$  will be assigned the correct value.

If it terminates because  $j = 0$ , then by the loop invariant we have  $\max S(i) \leq 1$ . If  $s_i = s_{j+1}(= s_1)$ , then  $1 \in S(i)$  so  $\max S(i) = 1$ , and since the increment of  $j$  in the **if** conditional runs,  $d_i$  is assigned the correct value. If  $s_i \neq s_{j+1}(= s_1)$ , then  $\max S(i) = 0$  and the increment doesn't run, so  $d_i$  is also assigned the correct value.

Therefore by induction,  $d_i = \max S(i)$  for all  $i = 1, 2, \dots, n$ , after the **for** loop terminates.

### Justification: correctness - loop 2

The second **for** loop simulates the behavior of DFA presented in (a). Since the correctness of the DFA is already proven in (a), we just show that this algorithm correctly simulates  $\delta$ . Here the role of variable  $i$  is to pinpoint the alphabet  $t_i$  the DFA is currently looking at, and  $j$  represents the DFA's current state  $q_j$ . Therefore we start with  $j = 0$  and  $i$  increases from 1 to  $m$ .

If  $j = n$  at the start of a **for** iteration, this means the current state is  $q_n$  and since  $\delta(q_n, c) = q_n$  for all  $c$ , we do nothing and stay at  $q_n$ . Otherwise, the behavior is simulated inside the **if** conditional.

If  $j < n$ , then  $\delta(q_j, t_i)$  is defined as  $q_{\max J(j, t_i)}$ . Therefore we must show that  $j$  updates to  $J(j, t_i)$  after the statements inside this conditional all run. Let  $j_0$  be the value of  $j$  at the start of **while** loop. We claim two loop invariants for the inner **while** loop:  $j + 1 \geq \max J(j_0, t_i)$  and  $s_1 \cdots s_j \leq s_1 \cdots s_{j_0}$ . Similarly to the first loop, we prove these by inducting on the loop count.

Initially  $j = j_0$ . If  $k \in J(j_0, t_i)$ , then  $s_1 \cdots s_k \leq s_1 \cdots s_{j_0} t_i$ , so  $k = |s_1 \cdots s_k| \leq |s_1 \cdots s_{j_0} t_i| = j_0 + 1$ , meaning  $j_0 + 1 \geq \max J(j_0, t_i)$  because it holds for all  $k \in J(j_0, t_i)$ . Also, obviously  $s_1 \cdots s_{j_0} \leq s_1 \cdots s_{j_0}$ .

Suppose that the invariants holds at the beginning of an iteration. Then we have  $j + 1 \geq \max J(j_0, t_i)$ , and since the loop did not terminate  $t_i \neq s_{j+1}$ . This means  $s_1 \cdots s_{j+1} \leq s_1 \cdots s_{j_0} t_i$  cannot hold and therefore  $j + 1 \notin J(j_0, t_i)$ , meaning  $j + 1 > \max J(j_0, t_i)$ . Suppose  $k \in J(j_0, t_i)$ . If  $k = 0$  then  $d_j + 1 \geq k$  is clear. Otherwise, we have  $s_1 \cdots s_k \leq s_1 \cdots s_{j_0} t_i \implies s_1 \cdots s_{k-1} \leq s_1 \cdots s_{j_0}$ , but since  $s_1 \cdots s_j \leq s_1 \cdots s_{j_0}$  and  $k - 1 \leq j$ ,  $s_1 \cdots s_{k-1} \leq s_1 \cdots s_j$ . Also  $k - 1 < j$  as  $k < j + 1$ , we have  $k - 1 \in S(j)$  by definition. Therefore,  $k - 1 \leq \max S(j) = d_j$ . Therefore,  $k \leq \max S(j) + 1$  holds for every  $k \in J(j_0, t_i)$ , meaning  $d_j + 1 = \max S(j) + 1 \geq \max J(j_0, t_i)$ . So updating  $j \leftarrow d_j$  preserves the first invariant. Also since  $d_j \in S(j)$ , we have  $s_1 \cdots s_{d_j} \leq s_1 \cdots s_j \leq s_1 \cdots s_{j_0}$ , the second invariant is also preserved.

Eventually the **while** loop must terminate because  $j$  is strictly decreasing at each iteration and  $j = 0$  breaks the loop. If it terminates because  $t_i = s_{j+1}$ , by the invariants we have  $j + 1 \geq \max J(j_0, t_i)$  and  $s_1 \cdots s_j \leq s_1 \cdots s_{j_0} \implies s_1 \cdots s_j s_{j+1} \leq s_1 \cdots s_{j_0} t_i \implies j + 1 \in J(j_0, t_i) \implies j + 1 \leq \max J(j_0, t_i)$ , so  $j + 1 = \max J(j_0, t_i)$ . In this case the increment of  $j$  inside the **if** conditional after the **while** loop runs, so  $j$  is updated to the correct value.

If it terminates because  $j = 0$ , then by the loop invariant we have  $\max J(j_0, t_i) \leq j + 1 = 1$ . If  $t_i = s_{j+1}(= s_1)$  then  $1 \in J(j_0, t_i)$  and therefore  $\max J(j_0, t_i) = 1$ , and the increment of  $j$  runs, so  $j$  is updated to the correct value. Otherwise  $\max J(j_0, t_i) = 0$  and the increment doesn't run so  $j$  retains its correct value of 0.

Therefore the second **for** loop correctly simulates the behavior of the DFA in (a). This means that after the loop terminates the DFA accepts  $t$  if and only if  $j = n$ , which is what the last two statements do to return the correct answer. Therefore, the given algorithm always terminates with the correct output.

### Justification: running time

All statements outside any loop runs in  $O(1)$  time (except for the list preparation step which may take  $O(n)$  time depending on the implementation).

In the first **for** loop, all statements outside the **while** loop run at most  $n$  times. Whenever the statement inside the **while** loop is run, the value of  $j$  strictly decreases, which means the total

number of times of the statement running is equal to the number of times where  $j$  decreases. Let's define that number  $n'$ . The value of  $j$  starts at 0, and can only increase by 1 at most  $n$  times, and decreases by at least 1  $n'$  times, so the final value of  $j \leq n - n'$ . But  $j$  never decreases below 0, so  $n - n' \geq 0 \implies n' \leq n$ . Therefore, the first loop uses  $O(n)$  operations.

In the second **for** loop, all statements outside the **while** loop run at most  $m$  times. Whenever the statement inside the **while** loop is run, the value of  $j$  strictly decreases. Let  $m'$  be the number of times that happens. Then  $j$  increases by 1 at most  $m$  times and decreases by at least 1  $m'$  times, the final value of  $j \leq m - m'$ , and since  $j$  never decreases below 0,  $m - m' \geq 0 \implies m' \leq m$ . Therefore the second loop uses  $O(m)$  operations.

Therefore the overall running time of this algorithm is  $O(n + m) = O(|s| + |t|)$ .  $\square$

(Remark from TA: There are several algorithms for this problem, and the most famous one among them is described here and called *Knuth–Morris–Pratt (KMP) algorithm*, which comes from automata theory. If you want to learn more about it, you may want to read Sections 32.3 (*String matching with finite automata*) and 32.4 (*The Knuth–Morris–Pratt algorithm*) of *Introduction to Algorithms* by Cormen et al, or the original paper *Fast Pattern Matching in Strings* by Knuth et al. Also note that it is okay even if you wrote an algorithm that works in time  $O(|s||\Sigma| + |t|)$  because we fixed  $\Sigma$ , which means we are assuming  $|\Sigma| = O(1)$  in this question. If you are unsure whether your algorithm is correct, you may want to implement the algorithm in a programming language and submit it to an online judge (e.g., BOJ 16916).)

2. Let  $k \in \mathbb{Z}^+$  be fixed and let  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ . Define  $L_k \subseteq \Sigma^*$  to be the language of all strings of length at least  $k$  whose  $k$ -th symbol from the end is  $\mathbf{b}$ . Equivalently,  $t \in L_k$  if and only if  $t = x\mathbf{b}y$  for some  $x, y \in \Sigma^*$  with  $|y| = k - 1$ .

- (a) Propose a DFA  $M$  recognizing  $L_k$ . Argue that  $L(M) = L_k$ .
- (b) Let  $P = \{\mathbf{a}, \mathbf{b}\}^k$ . Let  $M = (Q, \Sigma, \delta, q_0, F)$  be any DFA recognizing  $L_k$ , and let  $\hat{\delta}$  be the extended transition function of  $M$ . Define the function  $f: P \rightarrow Q$  by  $f(w) = \hat{\delta}(q_0, w)$ . Prove that  $f$  is injective. Conclude by arguing further that there is no DFA with fewer than  $2^k$  states that recognizes  $L_k$ .

*Solution.*

- (a) The idea is to design a deterministic finite automaton that recognizes  $L_k$  by remembering the *last  $k$  symbols seen*. For shorter prefixes, we use padding. Acceptance then depends only on whether, at the end of the input, the oldest symbol in this remembered  $k$ -window is  $\mathbf{b}$ . An explicit construction follows.

Let  $Q = \Sigma^k$  (all  $k$ -length words over  $\{\mathbf{a}, \mathbf{b}\}$ ). Interpret a state  $s = s_1 s_2 \cdots s_k \in Q$  as a “ $k$ -length window memory” holding the most recent  $k$  symbols, with  $s_1$  being the oldest and  $s_k$  the most recent.

- Alphabet:  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ .
- Start state:  $q_0 = \mathbf{a}^k$  (the word consisting of  $k$  copies of  $\mathbf{a}$ ). This serves as padding before  $k$  input symbols have been seen. Note that this padding cannot contain any  $\mathbf{b}$ , so this choice is the only consistent option.
- Transition: for  $s = s_1 \cdots s_k \in Q$  and  $\sigma \in \Sigma$ , define

$$\delta(s, \sigma) = s_2 s_3 \cdots s_k \sigma,$$

i.e., drop the oldest symbol and append the new one.

- Accepting states:

$$F = \{s_1 s_2 \cdots s_k \in Q \mid s_1 = \mathbf{b}\}.$$

Call this DFA  $M = (Q, \Sigma, \delta, q_0, F)$ . Note that  $|Q| = 2^k$ .

We immediately observe the following invariant of  $M$ : for any input  $w \in \Sigma^*$ ,

$$\hat{\delta}(q_0, w) = \text{the length-}k \text{ suffix of } (\underbrace{\mathbf{a} \cdots \mathbf{a}}_{k \text{ times}} w),$$

i.e., the last  $k$  symbols of  $w$  if  $|w| \geq k$ , and otherwise  $\mathbf{a}^{k-|w|}w$ .

It remains to verify that  $L(M) = L_k$ . Let  $t \in \Sigma^*$ .

- If  $|t| < k$ , then by the invariant the current state is  $\mathbf{a}^{k-|t|}t$ , whose first symbol is  $\mathbf{a}$ ; hence it is not in  $F$ , so  $t \notin L(M)$ . This matches  $t \notin L_k$  because the  $k$ -th symbol from the end is undefined.
- If  $|t| \geq k$ , then by the invariant  $\hat{\delta}(q_0, t)$  equals the last  $k$  symbols of  $t$ , say  $s_1 \cdots s_k$ . The first component  $s_1$  is exactly the  $k$ -th symbol from the end of  $t$ . Thus  $t$  is accepted if and only if  $s_1 = \mathbf{b}$ , i.e., if the  $k$ -th symbol from the end is  $\mathbf{b}$ . Therefore  $t \in L(M)$  if and only if  $t \in L_k$ .

Hence  $L(M) = L_k$ .

- (b) This subproblem was intended to foreshadow the kind of arguments we encounter in the study of DFA minimization and the Myhill–Nerode Theorem.

*Proof that  $f$  is injective.* Suppose that  $f$  is not injective, and that  $u, v \in \Sigma^k$  with  $u \neq v$ . Let  $i \in \{1, \dots, k\}$  be the smallest index such that  $u_i \neq v_i$ . Consider the continuation

$$y = \mathbf{a}^{k-i} \quad (k-i \text{ copies of } \mathbf{a}).$$

Then  $|y| = k - i$ , so in the concatenations  $uy$  and  $vy$ , the  $k$ -th symbol from the end is precisely the  $i$ -th symbol of  $u$  and  $v$ , respectively. Since  $u_i \neq v_i$  and  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ , exactly one of  $uy$  and  $vy$  lies in  $L_k$ .

If  $f(u) = f(v)$ , then from the same state the machine  $M$  would process the same suffix  $y$  and must either accept both  $uy$  and  $vy$  or reject both, contradicting the previous paragraph. Therefore  $f(u) \neq f(v)$ , and hence  $f$  is injective.  $\square$

To conclude, since  $f : P \rightarrow Q$  is injective and  $|P| = |\Sigma^k| = 2^k$ , we have  $|Q| \geq 2^k$ . Hence *every* DFA recognizing  $L_k$  has at least  $2^k$  states.

Part (a) provided a DFA with exactly  $2^k$  states recognizing  $L_k$ . Combined with this lower bound, we see that the minimal DFA for  $L_k$  has exactly  $2^k$  states.

As a concluding remark, we emphasize that by showing  $f$  is injective, we have effectively demonstrated that for any two distinct  $u, v \in \Sigma^k$ , it follows that  $u \not\equiv_{L_k} v$ , where the equivalence relation  $\equiv_{L_k}$  for  $L_k$  is defined in the same way as in the lecture slides.

$\square$

3. A set  $A \subseteq \mathbb{N}$  is *linear* if  $A = \{p + qn : n \in \mathbb{N}\}$  for some  $p, q \in \mathbb{N}$ . A subset of  $\mathbb{N}$  is *quasi-linear* if it is the union of finitely many linear sets.

- (a) Let  $A \subseteq \mathbb{N}$  be an arbitrary set. Prove that the set

$$A^\oplus := \{a_1 + \dots + a_k : k \in \mathbb{N}, a_1, \dots, a_k \in A\}$$

is quasi-linear.

- (b) Prove that a language  $L \subseteq \{a\}^*$  is regular if and only if the set  $\{n : a^n \in L\}$  is quasi-linear.

- (c) Prove that for any  $X \subseteq \{a\}^*$ , the language  $X^*$  is regular.

*Solution.*

- (a) We show that the set  $A^\oplus$  is quasi-linear for any  $A \subseteq \mathbb{N}$ . When  $A = \emptyset$  or  $A = \{0\}$ , then  $A^\oplus = \{0\}$  so it is trivially quasi-linear. Thus, suppose that  $A \neq \emptyset$ . Fix a positive element  $a \in A$ . For each  $i \in \{0, \dots, a-1\}$ , let  $A_i = \{x \in A : x \equiv i \pmod{a}\}$ . Then  $A = \bigcup_{i=0}^{a-1} A_i$ . Moreover, each nonempty  $A_i$  is linear since  $A_i = \{a_i + an : n \in \mathbb{N}\}$  where  $a_i$  is the smallest element in  $A_i$ . Therefore, we conclude that  $A^\oplus$  is quasi-linear.

- (b) Given a language  $L \subseteq \{a\}^*$ , let  $\text{Length}(L) := \{n : a^n \in L\}$ . The statement trivially holds when  $L = \emptyset$  or  $\text{Length}(L) = \emptyset$ , so assume both  $L$  and  $\text{Length}(L)$  are nonempty.

First, suppose that  $\text{Length}(L)$  is quasi-linear. Then  $\text{Length}(L) = A_1 \cup \dots \cup A_t$  where each  $A_1, \dots, A_t \subseteq \mathbb{N}$  is linear. For each  $i \in \{1, \dots, t\}$ , let  $A_i = \{p_i + q_i n : n \in \mathbb{N}\}$  and let  $L_i = \{a^{p_i + q_i n} : n \in \mathbb{N}\}$ . Observe that  $\text{Length}(L_i) = A_i$  and  $L = L_1 \cup \dots \cup L_t$ . Moreover, each language  $L_i$  is regular since it has a regular expression  $a^{p_i}(a^{q_i})^*$ . Thus, since  $L$  is a finite union of regular languages, the language  $L$  is also regular.

To show the converse, suppose that the language  $L$  is regular. Consider a DFA  $M = (Q, \{a\}, \delta, q_0, F)$  that recognizes  $L$  and let  $D$  be the transition diagram of  $M$ . Since  $L$  consists of exactly one symbol, each state has exactly one outgoing arc in  $D$ . In particular, this implies that  $D$  consists of a directed path starting from the initial state (called the *prefix part* of  $D$ ), followed by a directed cycle (called the *periodic part* of  $D$ ).

Let  $q_0, q_1, \dots, q_\ell$  and  $r_0 = q_\ell, r_1, \dots, r_{s-1}$  be the states in the prefix part and the periodic part in  $D$  that appear in this order. If  $q_i$  is the accepting state, then it recognizes the language  $\{a^i\}$ ; if  $r_j$  is the accepting state, then it recognizes the language  $\{a^{\ell+j+sn} : n \in \mathbb{N}\}$ . This shows that

$$\text{Length}(L) = \bigcup_{q_i \in F} \{i\} \cup \bigcup_{r_j \in F} \{\ell + j + sn : n \in \mathbb{N}\},$$

so we conclude that  $\text{Length}(L)$  is quasi-linear.

- (c) Take  $X \subseteq \{a\}^*$  and let  $L = X^*$ . Then  $\text{Length}(L) = \text{Length}(X)^\oplus$ , so  $\text{Length}(L)$  is quasi-linear by part (a). Therefore, by part (b), we conclude that  $L$  is regular.

□

4. Consider ONE-DIMENSIONAL MINESWEEPER (ODM for short) played on a board of shape  $1 \times n$  where  $n \in \mathbb{N}$ . Let  $\Sigma = \{0, 1, 2, \#\}$ . For  $w \in \Sigma^*$ , interpret  $w = a_1 a_2 \dots a_n$  as the configuration of the board, where the symbol  $a_i$  represents the  $i$ -th cell from the left on the board. The  $i$ -th cell and  $j$ -th cell are considered adjacent if and only if  $|i - j| = 1$ . A string  $w$  encodes a valid configuration if, for all  $i$  with  $1 \leq i \leq n$ , the following conditions hold:

- If  $a_i = \#$ , then the  $i$ th cell contains a mine.
- If  $a_i \in \{0, 1, 2\}$ , then the  $i$ th cell does not contain a mine, and the value of  $a_i$  equals the number of mines in its adjacent cells.

Then, consider CYCLIC ONE-DIMENSIONAL MINESWEEPER (CODM for short). The rules are the same as ODM, except that the first cell which represented by  $a_1$  and the last cell which represented by  $a_n$  are also considered as adjacent. Notice that '1#' is a valid configuration and '2#' is invalid configuration of CODM when  $n = 2$ .

Give a DFA  $M$  recognizing the language  $\{w \in \Sigma^* : w \text{ encodes a valid configuration of CODM}\}$ .

Hint: Start with find a DFA recognizing the language  $\{w \in \Sigma^* : w \text{ encodes a valid configuration of ODM}\}$ .

*Solution.*

- Method 1: Simple approach

Consider the case when the board is small. Assume that length of  $w$  is at most 2. Then, the accept cases are ' $\epsilon, 0, \#, 00, 1\#, \#1, \#\#$ '. Otherwise, the board forms invalid configuration of CODM, so DFA must reject it.

Let  $w = a_1 a_2 \dots a_n$  is a given board with  $n \geq 3$ . Then,  $w$  encodes a valid configuration of CODM if and only if every internal cell of  $w' = a_1 a_2 \dots a_n a_1 a_2$  follows the rule of ODM.

Using these two properties, we can design a DFA for this problem.

Let  $M = (Q, \Sigma, \delta, q_s, F)$  is DFA that we want. Each element of  $M$  is defined as following:

- $Q = \{q_\epsilon\} \cup \{q_a : a \in \Sigma\} \cup \{q_{ab} : a, b \in \Sigma\} \cup \{q_{abxy} : a, b, x, y \in \Sigma\} \cup \{q_{dead}\}$ .  
 $q_\epsilon$  denotes 'There is no cell that  $M$  read'.  
 $q_a$  denotes 'There is only one cell  $a_1$  that  $M$  read, and  $a_1 = a$ '.  
 $q_{ab}$  denotes 'There is two cells  $a_1, a_2$  that  $M$  read, and  $a_1 = a, a_2 = b$ '.  
 $q_{abxy}$  denotes 'There is at least 3 cells  $a_1, \dots, a_{k-1}, a_k$  that  $M$  read, and  $a_1 = a, a_2 = b, a_{k-1} = x, a_k = y$ '.  
 $q_{dead}$  denotes ' $M$  recognize / already recognized invalid cell'.
- $\Sigma = \{0, 1, 2, \#\}$  is given.
- Before define transition function, I will introduce a function  $f : \Sigma^3 \rightarrow \{\text{True}, \text{False}\}$ .  $f(x, y, z)$  is True if the middle cell  $y$  in the consecutive three cells  $x, y, z$ , does not violate the rule of ODM, and False otherwise. More precisely,  $f(x, y, z)$  is defined as following:

$$f(x, y, z) = \begin{cases} \text{True} & (y = 0, \text{both } x, z \text{ are number}) \\ & \vee (y = 1, x \text{ is number}, z \text{ is mine}) \\ & \vee (y = 1, x \text{ is mine}, z \text{ is number}) \\ & \vee (y = 2, \text{both } x, z \text{ are mine}) \\ & \vee (y \text{ is mine}) \\ \text{False} & \text{Otherwise} \end{cases}$$

Then,  $\delta : Q \times \Sigma \rightarrow Q$  is defined as following:

- \* For  $\forall z \in \Sigma, \delta(q_\epsilon, z) = q_z$ .
- \* For  $\forall a, z \in \Sigma, \delta(q_a, z) = q_{az}$ .
- \* For  $\forall a, b, z \in \Sigma, \delta(q_{ab}, z) = \begin{cases} q_{abbz} & \text{if } f(a, b, z) = \text{True} \\ q_{dead} & \text{if } f(a, b, z) = \text{False} \end{cases}$

$$* \text{ For } \forall a, b, x, y, z \in \Sigma, \delta(q_{abxy}, z) = \begin{cases} q_{abyz} & \text{if } f(x, y, z) = \text{True} \\ q_{dead} & \text{if } f(x, y, z) = \text{False} \end{cases}$$

$$* \text{ For } \forall z \in \Sigma, \delta(q_{dead}, z) = q_{dead}.$$

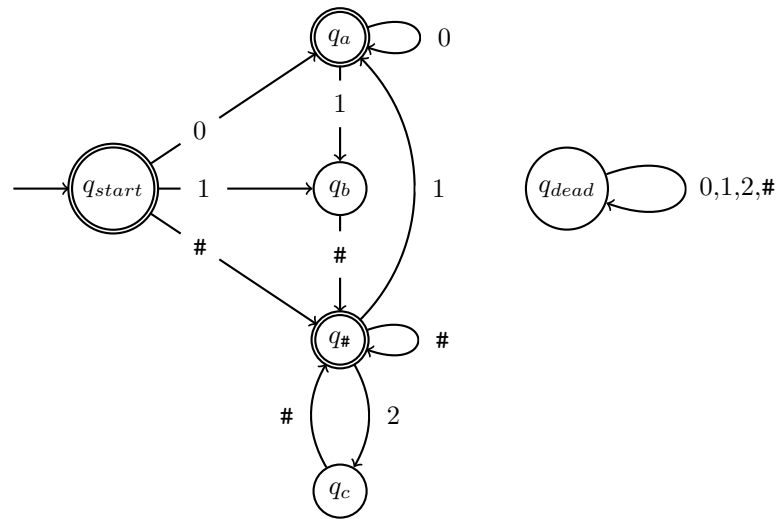
$$- q_s = q_\epsilon.$$

$$- F = \{q_\epsilon, q_0, q_\#, q_{00}, q_{1\#}, q_{\#1}, q_{\#\#}\} \cup \{q_{abxy} : \forall a, b \in \Sigma, f(x, y, a) = f(y, a, b) = \text{True}\}$$

- Method 2: Smaller transition diagram with less states.

We can construct a DFA using fewer states. In this approach, I will present only the transition diagram, since the explicit definition of the DFA has already been demonstrated in the previous method.

At first, DFA that recognizing valid configuration of ODM can be describe as following transition function. Notice that every edge leading to  $q_{dead}$  is omitted from the following transition diagram. (For example, edge that represents  $\delta(q_{start}, 2) = q_{dead}$  is not shown.)

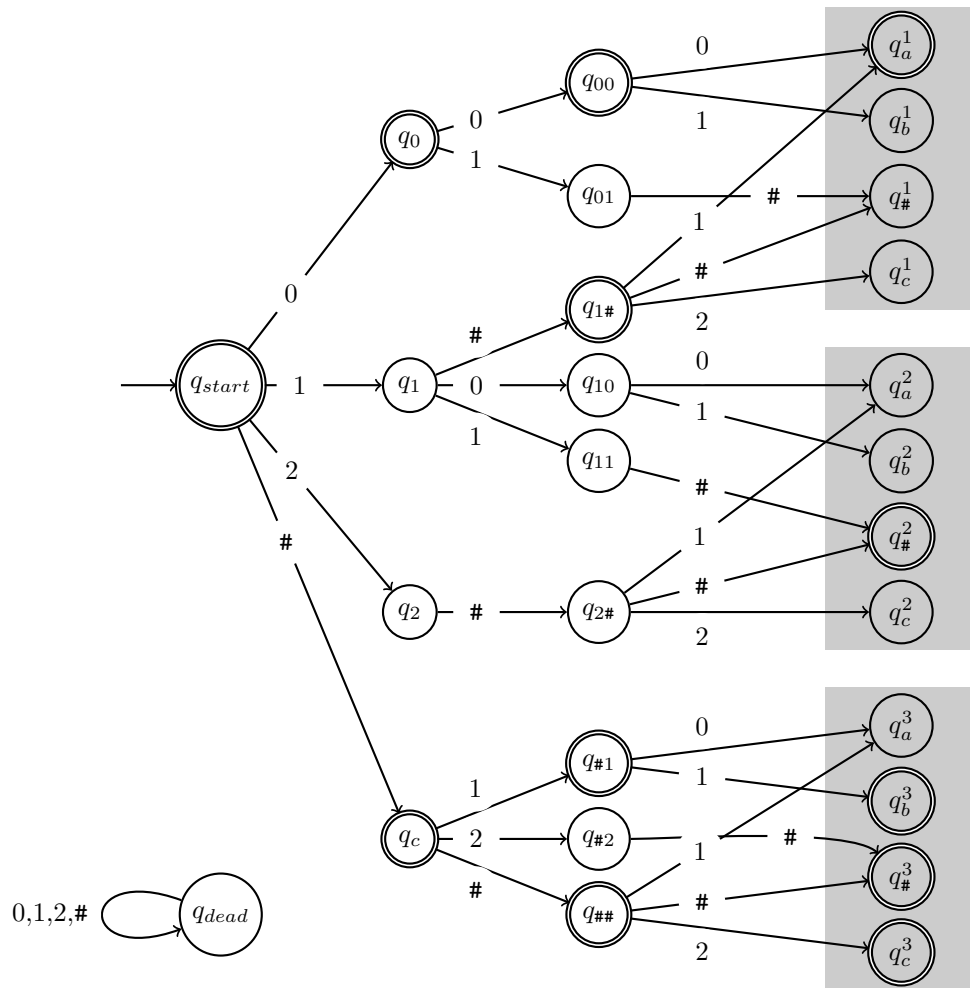


In above diagram,  $q_\#$  denotes that the current cell contains mine.  $q_a$  denotes that 'next cell should not contain mine'. Both  $q_b, q_c$  denote 'next cell should contain mine', but  $q_b$  also denotes 'current cell is 1', and  $q_c$  denotes 'current cell is 2'.

Then, consider DFA for CODM. Recall  $M = (Q, \Sigma, \delta, q_s, F)$  from upper method. Let  $Q^{ab} = \{q_{abxy} : x, y \in \Sigma\}$ , and  $F^{ab} = \{q_{abxy} : f(x, y, a) = f(y, a, b) = \text{True}\}$  for every  $a, b \in \Sigma$ . Notice that  $F = \{q_\epsilon, q_0, q_\#, q_{00}, q_{1\#}, q_{\#1}, q_{\#\#}\} \cup \bigcup_{a,b \in \Sigma} F^{ab}$ .

Then, fix arbitrary  $a, b \in \Sigma$ . Once  $M$  enters a state in  $Q^{ab}$ , it never transitions out of  $Q^{ab}$ . Moreover, the transitions among states in  $Q^{ab}$  are identical to those of the DFA for ODM. Therefore, each  $Q^{ab}$  in  $M$  can be replaced with a copy of the DFA for ODM.

Furthermore, we can observe that  $F^{00}, F^{01}, F^{1\#}$  are similar. It means that for  $x, y \in \Sigma$ ,  $q_{00xy} \in F^{00} \Leftrightarrow q_{01xy} \in F^{01} \Leftrightarrow q_{1\#xy} \in F^{1\#}$ . Likewise,  $F^{10}, F^{11}, F^{2\#}$  are similar and  $F^{\#1}, F^{\#2}, F^{\#\#}$  are similar. Using this observation, we can design a DFA  $M'$  for CODM as following. Notice that every edge leading to  $q_{dead}$  is omitted from the following transition diagram. And, the transitions among states in each gray box are identical to those in the diagram above for ODM, so they are also omitted from the following transition diagram.



□

5. Construct a DFA or an NFA recognizing the following languages.

- (a) Let  $\Sigma_1 = \{0, 1\}^3$  be an alphabet, where each symbol is written as columns. Construct an NFA  $M_1$  recognizing the following language  $L_1$ :

$$L_1 = \left\{ \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} \dots \begin{pmatrix} a_n \\ b_n \\ c_n \end{pmatrix} : [a_1 a_2 \dots a_n]_2 + [b_1 b_2 \dots b_n]_2 = [c_1 c_2 \dots c_n]_2 \right\}.$$

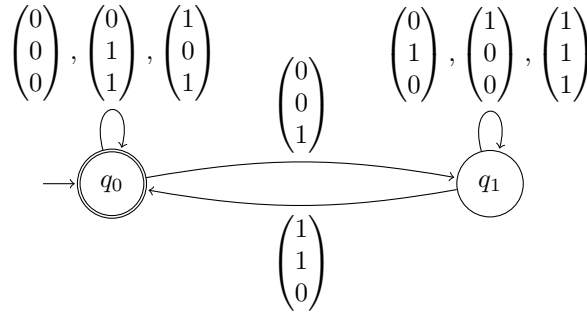
Here,  $[w]_2$  denotes the numerical value of the binary sequence  $w$ , e.g.,  $[101]_2 = 5$ . Along with an NFA, provide its transition diagram recognizing  $L_1$ .

- (b) Let  $\Sigma_2 = \{0, 1, \dots, b-1\}$  for  $b \geq 2$ . Construct a DFA  $M_2$  recognizing the base  $b$  representations of numbers divisible by  $k \in \mathbb{Z}^+$ .
- (c) Let  $\Sigma_3 = \{0, 1, \dots, b-1\}$  for  $b \geq 2$ . Construct a DFA  $M_3$  recognizing the reverse of the base  $b$  representations of numbers divisible by  $k \in \mathbb{Z}^+$ .

For (b) and (c), you may assume that the empty string  $\epsilon$  is 0, and the leading 0's of a base- $b$ -representation are allowed and does not affect its value, i.e., 0001 is equal to 1. Furthermore, you may provide transition functions for (b) and (c) if they are succinct and easy to understand.

*Solution.*

- (a) Let  $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ , where  $Q_1 = \{q_0, q_1\}$ ,  $F_1 = \{q_0\}$ , and  $\delta_1$  is given by the following transition diagram.



- (b) Let  $M_2 = (Q_2, \Sigma_2, \delta_2, 0, F_2)$ , where

$$\begin{aligned} Q_2 &= \{i : 0 \leq i \leq k-1\}, \\ \delta_2(q, a) &= b \cdot q + a \pmod{k} \text{ for } q \in Q_2, a \in \Sigma_2, \text{ and} \\ F_2 &= \{0\}. \end{aligned}$$

- (c) Let  $M_3 = (Q_3, \Sigma_3, \delta_3, (0, 1), F_3)$ , where

$$\begin{aligned} Q_3 &= \{(i, j) : 0 \leq i, j \leq k-1\}, \\ \delta_3((i, j), a) &= (i + j \cdot a \pmod{k}, j \cdot b \pmod{k}) \text{ for } (i, j) \in Q_3, a \in \Sigma_3, \text{ and} \\ F_3 &= \{(0, j) : 0 \leq j \leq k-1\}. \end{aligned}$$

Suppose we are given an input  $a_0 a_1 \dots a_n \in \Sigma^*$ . Then  $M_3$  has to accept if and only if

$$a_0 b^0 + a_1 b^1 + \dots + a_n b^n \pmod{k} = 0.$$

Say, we are at the state  $(i, j)$  after reading  $l$  symbols. The first coordinate  $i$  represents the current number modulo  $k$ , that is,  $a_0 b^0 + a_1 b^1 + \dots + a_l b^l \pmod{k}$ , and the second coordinate  $j$  represents  $b^{l+1} \pmod{k}$ . The start state is  $(0, 1)$ . After reading a symbol  $a_{l+1} \in \Sigma_3$ ,  $M_3$  moves to the state  $(x, y)$ , where

$$\begin{aligned} x &= a_0 b^0 + a_1 b^1 + \dots + a_l b^l + a_{l+1} b^{l+1} \pmod{k} = i + j \cdot a_{l+1} \pmod{k}, \text{ and} \\ y &= b^{l+2} \pmod{k} = j \cdot b \pmod{k}. \end{aligned}$$

The set of accept states consists of every state whose first coordinate is 0. □

6. A *monoid* is a pair  $(M, \circ)$  where  $M$  is a set and  $\circ : M \times M \rightarrow M$  is a binary operation on  $M$  that satisfies the following axioms:

(M1) The operation is associative, that is,  $a \circ (b \circ c) = (a \circ b) \circ c$  for every  $a, b, c \in M$ ; and

(M2) there is an element  $1_M \in M$  such that  $1_M \circ a = a \circ 1_M = a$  for every  $a \in M$ .

Given two monoids  $(M, \circ)$  and  $(N, \star)$ , a function  $h : (M, \circ) \rightarrow (N, \star)$  is a *monoid homomorphism* if  $h(a \circ b) = h(a) \star h(b)$  for every  $a, b \in M$  and  $h(1_M) = 1_N$ . Observe that for any set of alphabets  $\Sigma$ , the language  $\Sigma^*$  together with the concatenation operation forms a monoid. In this vein, we always assume that  $\Sigma^*$  is equipped with the concatenation operation.

Let  $\Sigma$  be an alphabet. Prove that  $L \subseteq \Sigma^*$  is regular if and only if there is a finite monoid  $(M, \circ)$  (i.e., a monoid where  $M$  is a finite set) and a monoid homomorphism  $h : \Sigma^* \rightarrow (M, \circ)$  satisfying the following.

There exists  $F \subseteq M$  such that for every  $w \in \Sigma^*$ ,  $w \in L$  if and only if  $h(w) \in F$ . (†)

*Solution.* Let  $L \subseteq \Sigma^*$ . First, suppose that there is a monoid  $(M, \circ)$ , a monoid homomorphism  $h : \Sigma^* \rightarrow (M, \circ)$ , and  $F \subseteq M$  that satisfy (†). Construct a DFA as follows: The set of states is  $M$ , the initial state is the identity element of  $(M, \circ)$ , the set of accepting states is  $F$ , and the transition function is defined by

$$(m, x) \mapsto m \circ h(x)$$

for  $m \in M$  and  $x \in \Sigma$ . Then it is routine to check that this DFA recognizes  $L$ .

To show the converse, suppose that the language  $L \subseteq \Sigma^*$  is regular. Let  $N = (Q, \Sigma, \delta, q_0, F)$  be a DFA that recognizes  $L$ . For each  $w \in \Sigma^*$ , let  $\varphi_w : Q \rightarrow Q$  be a function defined by  $\varphi_w(q) = \hat{\delta}(q, w)$ , where  $\hat{\delta}$  denotes the extended transition function. At last, let  $M = \{\varphi_w : w \in \Sigma^*\}$ .

Clearly,  $M$  is finite. Observe that  $M$  together with the composition operation  $\circ$  form a monoid, where the identity element is  $\varphi_\epsilon$ . Define a function  $h : \Sigma^* \rightarrow (M, \circ)$  by  $h(w) = \varphi_w$ . Then  $h$  is a monoid homomorphism. Moreover, if we let  $X := \{f \in M : f(q_0) \in F\}$ , then  $w \in L$  if and only if  $h(w) \in X$ . Therefore, we conclude that  $L$  is recognized by a finite monoid. □