# *Recitation 01: Finite Automata, Regular Languages*

In this recitation, we will review finite automata and see several examples of how to construct them to recognize specific languages. We will also review regular languages and regular operations, concluding with two methods of proving that all finite languages are regular.

## *Logistics*

Recitations will primarily be for reviewing lecture material and providing supplementary examples and exercises. They are optional, but if you find them useful, then you're encouraged to attend. Additionally, TAs will make note of attendance and participation, which may be used to improve low grades.

Recitation notes will be published shortly after each recitation; they may differ slightly from the content of your recitation due to variations in different instructor sections.

## *Finite Automata*

A *finite automaton* is a model of computation consisting of finitely many states and transitions. In particular, the states include exactly one *start state* and zero or more *accept states*. The strings inputted into a finite automaton consist of members of a specified alphabet, often $\{0, 1\}$. The formal definition of a *deterministic finite automaton* (or *DFA*), expressed as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, was described in Lecture 1.

We'll see several models of computation throughout the course; finite automata are among the simplest because the number of states is *finite*.

To review the formal definition of a finite automaton, see Definition 1.5 in the textbook.

The *language* of a finite automaton is the set of all strings that it accepts. Formally, a DFA $M = (Q, \Sigma, \delta, q_0, F)$ *accepts* a string $w = w_1 w_2 \cdots w_n$, where $w_i \in \Sigma$, if $\exists\, r_0, r_1, ..., r_n \in Q$ such that

1. $r_0 = q_0$
2. $r_i = \delta(r_{i-1}, w_i)$ for all $i = 1, ..., n$
3. $r_n \in F$.

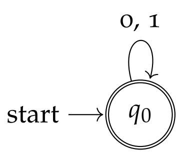
**Exercise 1.** *Construct a DFA that recognizes each of the following languages, where* $\Sigma = \{0, 1\}$*:*

1. $\Sigma^*$
2. $\emptyset$
3. $\{\varepsilon\}$

Remember that $\Sigma^*$ is the set of all strings consisting of zero or more 0s and 1s.

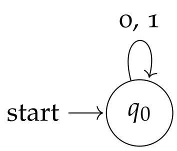
**Solution.** We construct DFAs for each of the three languages.

1. $\Sigma^*$



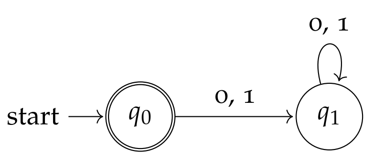

The only state is both the start state and the only accept state. This means that any string that enters the DFA will be accepted!

2. $\emptyset$




A small modification from the above: now no strings should be accepted, so we simply remove the accept state.

3. $\{\varepsilon\}$



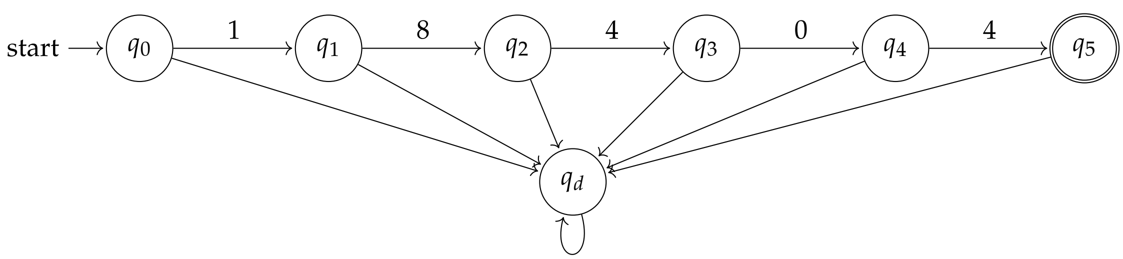

All strings consisting of at least one 0 or 1 will lead us to $q_1$ and never return to $q_0$, so they will be rejected.

**Exercise 2.** *Construct a DFA that recognizes each of the following languages with the given* $\Sigma$*:*

1. $\Sigma = \{0, 1, ..., 9\}$, $A_{18404} = \{18404\}$.
2. $\Sigma = \{a, b\}$, $A_{even} = \{w \mid w$ *has an even number of a's*$\}$.

**Solution.** We construct DFAs for each of the two languages.

1. $\{18404\}$

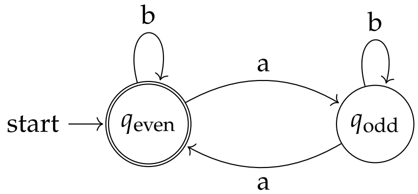



Here, $q_d$ represents the "*dead state*." Once we land in this state, we will never leave, so the input will not be accepted.

An arrow with no label implies that all alphabet members not included in any other transition from that state will follow this arrow. For example, from $\delta(q_2, n) = q_d$ for all $n \neq 4$.

Note that $q_1$ in Exercise 1.3 is also a dead state!

2. $\{w \mid w$ has an even number of $a$'s$\}$




Being in $q_{\text{even}}$ represents that we have read in an even number of $a$'s; being in $q_{\text{odd}}$ represents that we have read in an odd number of $a$'s.

When constructing DFAs, it's useful to think of each state as a *possibility* for the value that the machine cares about. In Exercise 2.2 above for example, the value that the machine cares about is whether there is an even or odd number of $a$'s so far: in other words, the *parity* of the number of $a$'s. Since there are two possible parities, we have two states.

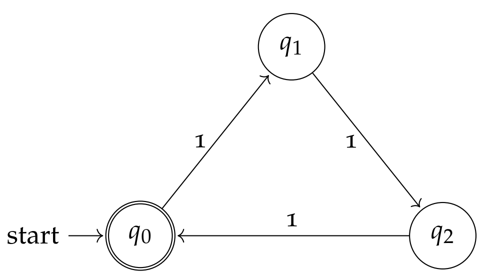
**Exercise 3.** *With $\Sigma = \{1\}$ (unary strings), construct a DFA that recognizes $A_3 = \{w \mid w$ has length a multiple of 3$\}$. Then, by generalizing this idea, provide a formal definition of a DFA recognizing $A_n = \{w \mid w$ has length a multiple of $n\}$.*

*Hint.* What is the value that the machine needs to keep track of? How many possibilities are there for that value?

**Solution.** We know that the DFA should keep track of a value related to the length of the input; however, we don't care about the actual length as much as whether it is divisible by 3. Thus, we create three states to represent the *length modulo 3* of the input so far.

"Length modulo 3" is the remainder when we divide the length by 3.

We start at $q_0$, representing that the current length is 0 mod 3. Note that this is also the accept state, since we want to accept strings with length 0 mod 3. On the next 1, the length becomes 1 mod 3, then 2 mod 3, then back to 0 mod 3, etc.




For arbitrary $n$, we want a cycle like the above, but consisting of $n$ states, representing the length being $0, 1, ..., n-1$ modulo $n$. $M = (Q, \Sigma, \delta, q_0, F)$ recognizes $A_n$, where

$$\begin{aligned} Q &= \{q_0, ..., q_{n-1}\} \\ \Sigma &= \{1\} \\ \delta(q_i, 1) &= \begin{cases} q_0 & i = n-1 \\ q_{i+1} & \text{else} \end{cases} \\ q_0 &= q_0 \\ F &= \{q_0\} \end{aligned}$$

## *Regular Languages*

A *regular language* is any set of strings that is recognized by some finite automaton.

Let $L_1 = \{a, b\}$, $L_2 = \{b, c\}$. In Lecture 1, we defined the following *regular operations*:

- Union ($\cup$)

  Example: $L_1 \cup L_2 = \{a, b, c\}$.

- Concatenation ($\circ$)

  Example: $L_1 \circ L_2 = L_1 L_2 = \{ab, ac, bb, bc\}$.

- Star ($*$)

  Examples: $\emptyset^* = \{\varepsilon\}$, $L_1^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\}$.

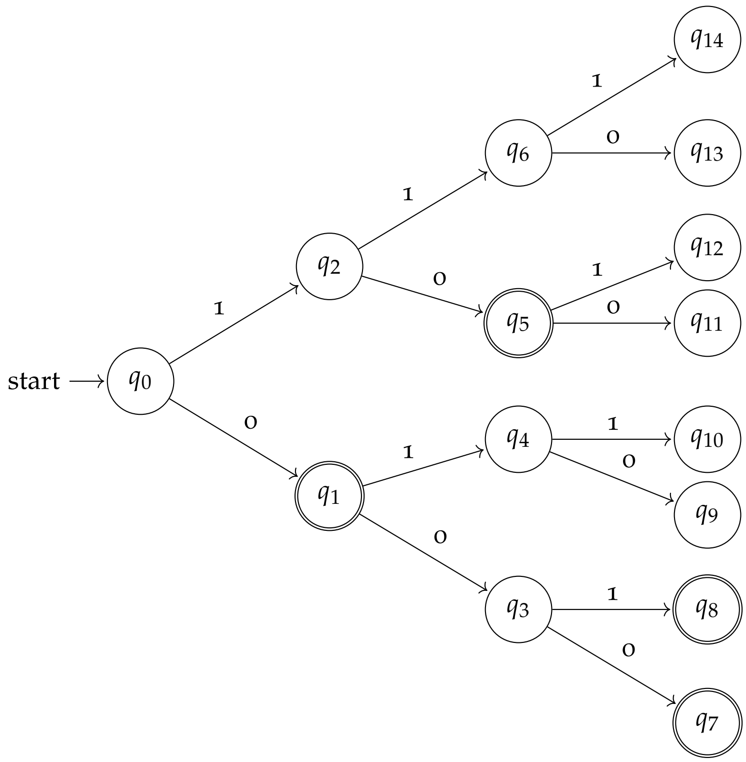
**Theorem 1.** *Every finite language is regular.*

*Proof 1 (construction).* Let $L = \{w_1, w_2, ..., w_n\}$ be a finite language. Define $k$ to be the maximum length of a string in $L$, or $k = \max_{1 \le i \le n} |w_i|$. Construct a tree of depth $k + 1$, where each state splits into two new states (one for 0 and one for 1). See the example below for concreteness. Note that there is now a unique path for every string of length $k$.

For each $w_i \in L$, add an accept state corresponding to the state reached after processing $w_i$. This ensures that strings lead to an accept state if and only if the string is a member of $L$. Thus, this construction recognizes $L$, which shows that $L$ is regular. □

Note that it would not be possible to find $k$ for an infinite language, so this construction only applies to finite languages.

For example, the following is the DFA for $L = \{0, 10, 000, 001\}$ using this construction.




In additiona, there is an dead state from which states $q_7, ..., q_{14}$ lead to on any input, since we don't want to accept any strings of length longer than 3. We haven't drawn the dead state or its associated transitions to simplify the diagram.

*Proof 2 (closure under union).* Again let $L = \{w_1, w_2, ..., w_n\}$ be a finite language. Let $L_i := \{w_i\}$ for $i = 1, ..., n$. For each $i$, we can construct a DFA that recognizes $L_i$ in the same manner as Exercise 2.1 ($A_{18404}$). Thus, $L_i$ is a regular language. In Lecture 1, we showed that regular languages are closed under union. We know that $L = L_1 \cup L_2 \cup \cdots \cup L_n$, so $L$ is also regular. $\square$