# COM364 Automata Theory
# Lecture Note[*]1 - Finite Automata

Kurtuluş Küllü

March 2018

Figure 1 shows a finite automaton (FA) (automaton is singular and automata is plural) that we will call $M_1$. Such a drawing is called the *state diagram* of $M_1$. It has three states: $q_1$, $q_2$, and $q_3$. The *start state*, $q_1$, is indicated by the arrow pointing at it from nowhere. The *accepting (final) state*, $q_2$, is drawn with double circles. A FA always has only one starting state but it can have multiple accepting states. Arrows going from one state to another show the *transitions*.

When this automaton receives an input string such as 1101, it processes the string symbol by symbol and produces only one output: accept/reject. This output depends on which state the machine is at the end of the input string.

$$q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{1} q_2.\text{accept}.$$

$M_1$ accepts strings such as 1, 01, 11, 01010101, ... (any string that ends with a 1) and 100, 0100, 110000, 01010000, ... (strings ending with even number of 0s after the last 1). It rejects other strings (such as 0, 10, 101000, ...).

## Formal Definition of FA

Formally, a FA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set of states,
2. $\Sigma$ is the input alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accepting (final) states.

[*]Based on the book "Introduction to the Theory of Computation" by Michael Sipser.

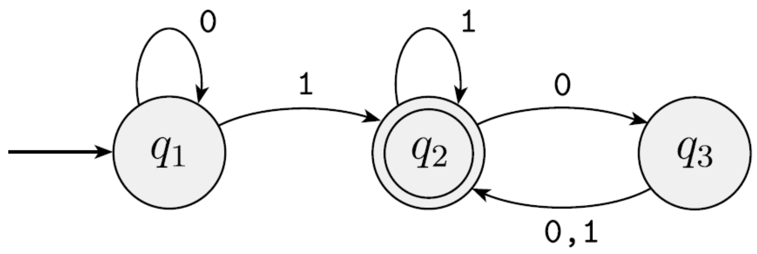

Figure 1: The first FA example.

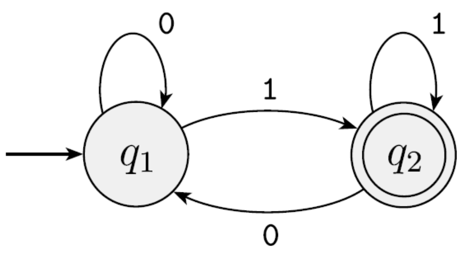

Figure 2: The state diagram of $M_2$.

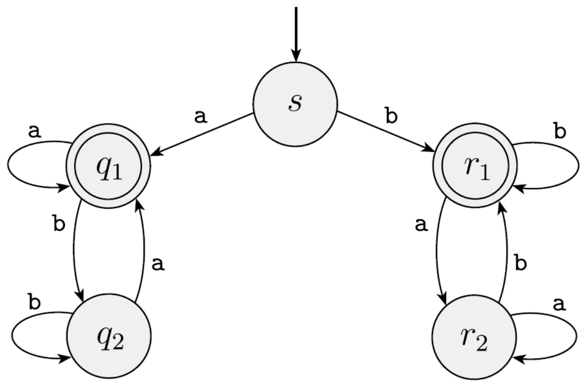

Figure 3: The state diagram of $M_4$.

For example, for the first example $M_1$, we can write

$$M_1 = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_2\})$$

where $\delta$ is given with the following table.

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_2$ | $q_2$ |

If $A$ is the set of all strings that a FA, $M$, accepts, we say that $A$ is the *language of machine $M$* (or that "$M$ recognizes $A$") and we write $L(M) = A$.

**Example:** $M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$ where $\delta$ is

| | 0 | 1 |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_1$ | $q_2$ |

The state diagram of $M_2$ is given in Figure 2. Note that the formal definition and state diagram contain the same information and that one can be obtained from the other.

**Exercises:** Work on your own and think about how $M_2$ processes an input string like 1101. Can you find out the language recognized by $M_2$? In other words, what is the set $A_2 = L(M_2)$? What changes if we create a new machine $M_3$ from $M_2$ by making $q_1$ a final state and $q_2$ a non-final state?

**Note:** If the starting state is also an accepting state, then the FA accepts the *empty string* ($\varepsilon$).

**Example:** The state diagram for the FA $M_4$ is given in Figure 3. This automaton is the first example where we see more than one accepting state. Also, you should note that the input alphabet is different. The strings for this machine consists of $a$'s and $b$'s. Additionally, the start state of $M_4$ is only used at the beginning. After processing the first symbol of the input string, the automaton never goes back to state $s$. Because of this and the fact that there is no passage from $q$ states to $r$ states or in the opposite direction, it can be considered like two automata combined together.

If the input string starts with an $a$, we go left to $q$ states, which makes sure that the string ends with an $a$. Otherwise (i.e., if the first symbol is a $b$), we go right to $r$ states, which makes sure that the string ends with a $b$. Combined together, they all work to make sure that the input string begins and ends with the same symbol.

## Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a FA and let $w = w_1 w_2 \dots w_n$ be a string where each $w_i \in \Sigma$. $M$ *accepts* $w$ if a sequence of states $r_0, r_1, \dots, r_n$ in $Q$ exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, 1, \dots, n-1$, and
3. $r_n \in F$.

$M$ *recognizes* language $A$ if $A = \{w \mid M \text{ accepts } w\}$.

**Definition:** A language is called a "*regular language*" if some FA recognizes it.

## Designing a FA for a language/problem

Like proving a theorem, this process involves creativity. So, there is no algorithm (a recipe/formula) for designing a FA when given a language. But, there are useful techniques. For example:

- Put yourself in place of the automaton;
- Work on some example strings;
- Think about what you/the automaton need(s) to remember when looking at the next symbol at any point (**Question:** Why not remember all?)

**Note:** Keep in mind that you/the automaton sees the string symbol by symbol. After each symbol, the decision for the string up to that point must be correct because when the end of the string comes is unknown.

**Example:** Design a FA that recognizes the language

$$A = \{x01y \mid x \text{ and } y \text{ are any strings of 0s and 1s}\}$$

.

Some example strings in $A$ are 01, 11010, and 100011. Some example strings not in $A$ are $\varepsilon$, 0, 11000. For the formal definition, what do we already know about this machine? We know that input alphabet is binary, $\Sigma = \{0, 1\}$. (Actually, this is the smallest set. Any set including 0 and 1 will work.) It will have states ($Q$), one of which, $q_0$, will be the start state. What does it have to remember at any point when processing a string? Have we seen a 01?

- If so, the string will be accepted.
- If not,
  - and if the last symbol was a 1 or if we are looking at the first symbol, we have not seen anything towards a 01.
  - and if the last symbol was a 0, the next symbol is perhaps a 1 which should cause the string to be accepted. So, this should be a separate situation that we should remember.

The FA for this language is drawn in Figure 4. The formal definition can be completed from this diagram. Think about how each state corresponds to the different situations described in the items above.

**Exercise:** $\Sigma = \{0, 1\}$, design a FA that recognizes the language that contains all strings with odd number of 1s.

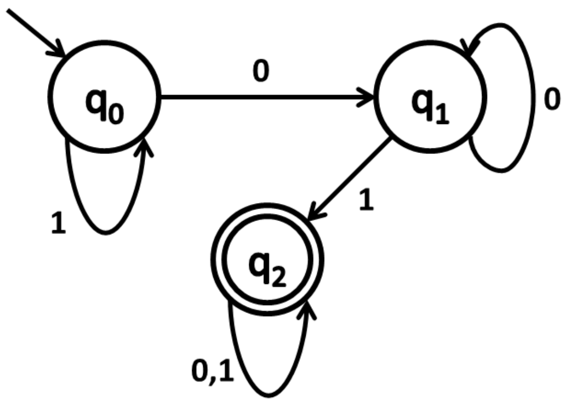

Figure 4: The FA for the first design example.

## Properties of FA and Regular Languages

Important operations on regular languages are *set operations* (union, intersection, complement, difference), *concatenation*, and *star*. A language is also a set, so the classical set operations such as union and intersection can be applied to them. The result of applying these operations to languages can again be considered as a language. We won't redefine these operations here. The other two operations are defined as below.

- Concatenation of languages $A$ and $B$ is $AB = \{xy \mid x \in A \text{ and } y \in B\}$.
- Star is a unary operation and defined as $A^* = \{x_1x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

**Note:** For any $A$, always $\varepsilon \in A$.
**Example:** Let $\Sigma = \{a, b, c, \dots, z\}$. If $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{boy}, \text{girl}\}$, then

$$\begin{aligned} A \cup B &= \{\text{good}, \text{bad}, \text{boy}, \text{girl}\} \\ AB &= \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\} \\ A^* &= \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \\ &\quad \text{goodgoodgood}, \dots, \text{badbadbad}, \dots\} \end{aligned}$$

### Closure Properties

In general, a collection of objects is closed under some operation if applying that operation to members of the collection returns an object also in the collection. For example, $\mathbb{N}$ is closed under multiplication but it is not closed under division.

The regular languages are closed under set operations, concatenation, and star operation. Because of this, they are sometimes called *regular operations*.

**Theorem:** The set of regular languages is closed under the union operation. In other words, if $A$ and $B$ are regular languages, then so is $A \cup B$.

**Proof Idea:** Because $A$ and $B$ are regular, (according to the definition of regular languages) there has to be machines that recognize them. Let us call them $M_A$ and $M_B$ respectively. We can use proof by construction and show that a machine $M$ can be constructed from $M_A$ and $M_B$ and that $M$ recognizes $A \cup B$.

$M$ must accept a string exactly when $M_A$ or $M_B$ would accept it. So, in a way, $M$ needs to simulate both $M_A$ and $M_B$. How can this be done? If we try to run one of the machines and then the other one, we need to rewind (go back to the beginning of) the input string in between. This is not possible. So, both $M_A$ and $M_B$ should be simulated simultaneously.

To do this, $M$ needs to remember, at any time, both the states that $M_A$ and $M_B$ are in. If $M_A$ has $k_A$ states and $M_B$ has $k_B$ states, there are $k_A \times k_B$ possible pairs to remember and these are

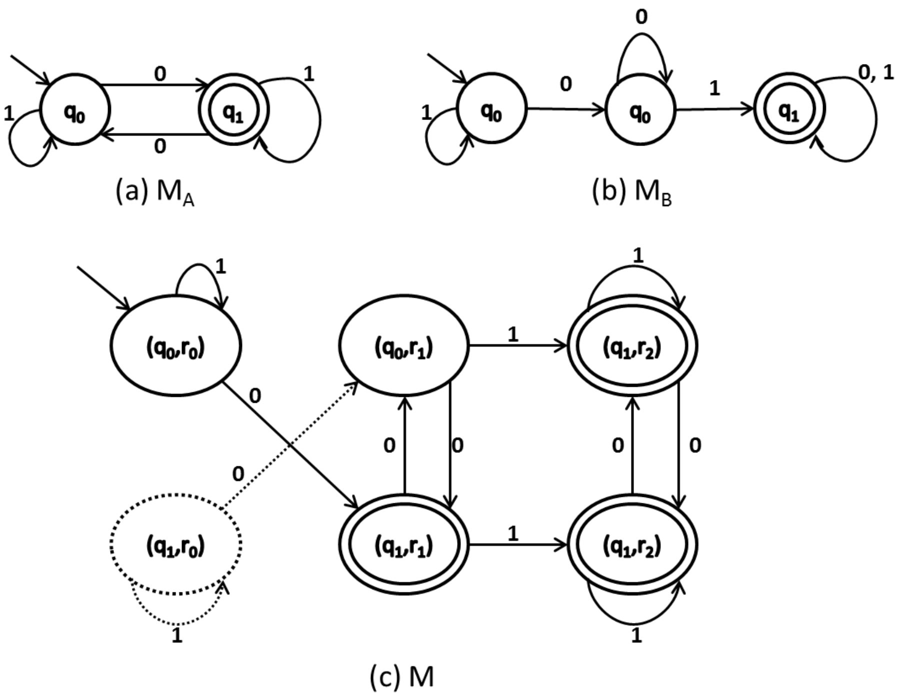

Figure 5: (a) The FA that recognizes $A$. (b) The FA that recognizes $B$. (c) The FA constructed from $M_A$ and $M_B$ that recognizes $A \cup B$.

our states for $M$. The start state of $M$ will be the pair consisting of the start states of $M_A$ and $M_B$. Transitions for $M$ can be found out from the transitions of $M_A$ and $M_B$. And, because we want to create a machine for the union, we want to accept an input string when one of $M_A$ or $M_B$ accepts it. Therefore, the final states of $M$ will be those that include a final state of $M_A$ or $M_B$.

**Proof:** Let $M_A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ be the automaton that recognizes $A$ and $M_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ be the automaton that recognizes $B$. (**Note:** We take the alphabets to be the same ($\Sigma$) to make the proof simpler. It is also possible to prove when alphabets are different.)

We can construct an automaton $M = (Q, \Sigma, \delta, q_0, F)$ that will recognize $A \cup B$ as follows.

1. $Q = \{(r_A, r_B) \mid r_A \in Q_A \text{ and } r_B \in Q_B\}$. So, $Q = Q_A \times Q_B$.
2. $q_0 = (q_A, q_B)$. The start state is the pair formed with both start states.
3. $F = \{(r_A, r_B) \mid r_A \in F_A \text{ or } r_B \in F_B\}$. (One of the pair is an accepting state.)
4. $\delta$ can be defined as follows. For each $(r_A, r_B) \in Q$ and each $x \in \Sigma$, $\delta((r_A, r_B), x) = (\delta_A(r_A, x), \delta_B(r_B, x))$.

This concludes the construction. It is a simple construction and with a little effort, the reader can be convinced that $M$ recognizes $A \cup B$. If not, or for more complex constructions in other proofs, it will be necessary to prove also that the constructed $M$ accepts only strings in $A \cup B$. (For the current construction, induction can be used.)

**Example:** $A = \{x \mid x \text{ has odd 0s}\}$, $B = \{x \mid x \text{ has 01 in it}\}$. The automata $M_A$ and $M_B$ that recognize languages $A$ and $B$ and the automaton $M$ constructed from $M_A$ and $M_B$ to recognize $A \cup B$ as described above are all shown in Figure 5. The reason for showing one of the states of $M$ and its transitions dashed is that there is no way to reach that state from the start state. So, the dashed parts can actually be omitted without changing the machine.

**Question:** How can the proof for union operation be adapted to prove that the set of regular languages is closed under intersection operation?

**Theorem:** The set of regular languages is closed under the concatenation operation. In other words, if $A$ and $B$ are regular languages, then so is $AB$.

**Note:** For an easy proof of this theorem, we need to define a new idea called *nondeterminism* (see next lecture notes).