Finite Automata

1 Finite Automata

Formally, a finite automata, or finite state machine is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where

1. \(Q\) is a finite set called the STATES, denoted as blobs.
2. \(\Sigma\) is a finite set (corresponding to the input) called the alphabet.
3. \(\sigma : Q \times \Sigma \to Q\) is the transition function. This is denoted as arrows between the blobs, marked by the corresponding alphabet.
4. \(q_0 \in Q\) is the starting state, denoted by an arrow entering it from nowhere.
5. \(F \subseteq Q\) is the set of accept states, denoted by a double circle.

As a refresher, the text book introduces automatas via an example with three states, \(Q = \{q_1, q_2, q_3\}\), and a binary alphabet \(\Sigma = \{0, 1\}\). The starting state is \(q_1\), and there is only one accepting state \(q_2\). The transitions are:

<table>
<thead>
<tr>
<th>(Q \setminus \Sigma)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>(q_1)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_3)</td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_2)</td>
<td>(q_2)</td>
</tr>
</tbody>
</table>
We can check that this machine accepts 1101, as well as 1, 01, 0101010101, but does not accept the strings 0, 10, 1010000.

With some careful reasoning, we can check that for this machine $M$, we have

$$L(M) = \{w : w \text{ contains at least one 1, and the last 1 is followed by an even number of 0s}\}.$$

There are two ‘corner cases’ worth mentioning:

1. We use $\epsilon$ (epsilon) to denote the empty string. It’s possible for a machine $M$ to have $\epsilon \in L(M)$: we just need $q_0 \in F$.

2. It’s also possible for $M$ to accept no strings. For example, $F$ can simply be empty. In such cases, we formally have $L(M) = \emptyset$.

A language that’s accepted by some finite automata is known as a regular language.

### 1.1 Binary Strings Divisible by 3

Note that binary strings also represent numbers: the leftmost character is the most significant, the rightmost is least significant.

We will use $\overline{w}$ to represent the number corresponding to the string/word $w$.

We can create an automata using the following observation: if we attach a character $x$ onto the end of a word $w$, we get

$$\overline{wx} = 2\overline{w} + \overline{x}.$$  

That is, the remainder of $\overline{wx}$ when divided by 3 is determined by the remainder of $\overline{w}$ divided by 3, along with the value of $x$. Note that the operations (multiplication and addition) are actually manipulating numbers, instead of strings. This is because $\overline{w}$ and $\overline{wx}$ are already numbers.

Specifically, we get the following table:

<table>
<thead>
<tr>
<th>$\overline{w}$ mod 3</th>
<th>$\overline{w0}$ mod 3</th>
<th>$\overline{w1}$ mod 3</th>
<th>$\overline{w2}$ mod 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

This is exactly the property we want from an automata: the state of the new word is given precisely by the state of the previous word, and the new character added. We create states

$$Q = \{q_0, q_1, q_2\}$$

to correspond to the case where the string read so far having residue 0, 1, and 2 modulo 3 respectively. Then

- because the empty string corresponds to 0, we start at $q_0$, and
because we want to only accept the strings that correspond to numbers with residue 0 modulo 3, we set $F = \{q_0\}$.

The transitions are almost identical to the table above, except we replace explicit modulos with states

<table>
<thead>
<tr>
<th>$Q \setminus \Sigma$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

1.2 From (Sets of) Strings to Automatas

We can also go the other way: we can go from any set of strings to an automata that accepts them. For example, suppose we’re in the binary alphabet $\Sigma = \{0, 1\}$ again, and have

$$A = \{0, 001\},$$

then an automata that works is one with five states $Q = \{q_1, q_2, q_3, q_4, bad\}$, where we start at $q_1$, and have transitions:

<table>
<thead>
<tr>
<th>$Q \setminus \Sigma$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
<td>bad</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>bad</td>
</tr>
<tr>
<td>$q_3$</td>
<td>bad</td>
<td>$q_4$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>bad</td>
<td>bad</td>
</tr>
<tr>
<td>bad</td>
<td>bad</td>
<td>bad</td>
</tr>
</tbody>
</table>

that is, we create a new state whenever we’re still in the prefix of one of the strings in $A$, and accept when we hit exactly one of these strings. That is, we need to start at $q_1$, and have

$$F = \{q_2, q_4\}.$$

Formally, this type of logic shows

**Lemma 1.1.** For any finite set $A$, there is a finite automaton $M$ such that

$$L(M) = A.$$
2. Arrows can also be labeled with $\epsilon$, which means it does not consume a character of the input, and makes the move.

As an example, consider an automata for accepting strings ending with 01. This is the concatenation of the language of all strings, along with the language containing just 11:

$$\left(\{0, 1\}^*\right) \circ \{11\}$$

A DFA $M_1$ for $\{0, 1\}^*$ can just has one state, $q_0$, which is accepting, and both transitions going back to it.

A DFA $M_2$ for $\{11\}$ has four states, $q_1$, $q_2$, $q_3$, and $bad = q_4$. $q_3$ is accepting, and the transitions are:

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$\Sigma$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
<td>$q_4$</td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_4$</td>
<td>$q_3$</td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_4$</td>
<td>$q_4$</td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td>$q_4$</td>
<td>$q_4$</td>
<td></td>
</tr>
</tbody>
</table>

To concatenate these, we simply add an $\epsilon$ transition from $q_0$ to $q_1$: that is, we can switch over from $M_1$ to $M_2$ at any point.

It’s useful to consider how this NFA operates on the input 011101: it essentially ‘branches’ into multiple states at each point of the input.

We formally define an NFA as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set of states.
2. $\Sigma$ is a finite alphabet.
3. $\delta : Q \times (\Sigma \cup \epsilon) \mapsto \mathcal{P}(Q)$ is the transition function ($\mathcal{P}(Q)$ is the power-set of $Q$).
4. $q_0 \in Q$ is the starting state.
5. $F \subseteq Q$ is the set of accept states.

Note that the only difference from a DFA is in $\sigma$. The first difference is that the transitions can accept $\epsilon$. We sometimes refer to the $\epsilon$-inclusive alphabet as $\Sigma_\epsilon$. Secondly, each transition can now lead to multiple states.

We say that an NFA $N = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w$ if

- we can write $w$ as $w = y_1 y_2 \cdots y_m$, where each $y_i \in \Sigma_\epsilon$;
- there exists a sequence of states $r_0 r_1 \cdots r_m$ such that
  - $r_0 = q_0$
  - $r_m \in F$
  - $r_{i+1} \in \delta(r_i, y_{i+1})$
2.1 Example 1.

Consider the NFA $N_1$:

Formal definition: $N_1 = (Q, \Sigma, \delta, q_0, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. $\delta =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>${}$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>${}$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>${}$</td>
<td>${q_4}$</td>
<td>${}$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_1}$</td>
<td>${q_4}$</td>
<td>${}$</td>
</tr>
</tbody>
</table>

4. $q_0 = q_1$
5. $F = \{q_4\}$

Question: What is $L(N_1)$?

First of all let us check if a string, say, 11 is accepted by $N_1$. The possible paths taken by the automaton on input 11 are:

- $q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1$
- $q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_2$
- $q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{\epsilon} q_3$
- $q_1 \xrightarrow{1} q_2 \xrightarrow{\epsilon} q_3 \xrightarrow{1} q_4$

The last path here is an accept path, hence $11 \in L(N_1)$.

Let us check similarly for 10:

- $q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_1$
- $q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3$

None of these are accept paths, hence $10 \notin L(N_1)$.

Observing carefully, we see that any path from $q_1$ to $q_4$ must consume a substring 11 or 101. And since $q_1$ and $q_4$ contain self-loops with both 0 and 1, all strings are allowed on either side. Hence, $L(N_1)$ is the set of all strings containing either 11 or 101 as a substring.
2.2 Example 2

Let $L$ be a language consisting of all strings over \{0, 1\} such that the third-from-last digit is 1.

To do this, we can use this idea: first we accept any string, then we accept a 1, and then 2 more input characters which could be 0/1.

This gives the following NFA:

$$
\begin{array}{c}
\text{start} \\
\circlearrowright
\end{array}
\begin{array}{c}
q_1 \\
\downarrow 0, 1
\end{array}
\begin{array}{c}
q_2 \\
\rightarrow 0, 1
\end{array}
\begin{array}{c}
q_3 \\
\rightarrow 0, 1
\end{array}
\begin{array}{c}
q_4
\end{array}
$$

Let us do a sanity check with a couple of strings in the language, to check if each of them has at least one accept path in $N_2$.

- String: 01101. Accept path: $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{1} q_4$
- String: 1111. Accept path: $q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_4$

Now consider the string 1001. It’s possible paths are:

- $q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2$
- $q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3 \xrightarrow{0} q_4 \xrightarrow{1} ???$

None of these are accepting paths, so our NFA checks out in this case.

3 From NFAs to DFAs

The most important fact about NFAs is that the languages accepted by them is precisely the regular languages.

A DFA is a NFA in that there are no $\epsilon$ transitions, and all transitions go to exactly one state.

So we need to show that any NFA has an equivalent DFA that accepts the same states. Recall that an NFA $N = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w$ if

- we can write $w$ as $w = y_1y_2 \cdots y_m$, where each $y_i \in \Sigma_i$;
- there exists a sequence of states $r_0r_1 \cdots r_m$ such that
  - $r_0 = q_0$
  - $r_m \in F$
  - $r_{i+1} \in \delta(r_i, y_{i+1})$
The way that we check whether a string is accepted is similar to reachability check on graphs: we create a set of all reachable states after we have examined \( y_1 \ldots y_i \). That is, we successively update

\[
R_{i+1} \leftarrow \bigcup_{r \in R_i} \delta(r, y_{i+1}).
\]

For example, for the NFA with two states \( Q = \{q_0, q_1\} \), \( F = \{q_0\} \), \( \Sigma = \{0, 1\} \), and transitions

\[
\begin{array}{c|cc}
Q \setminus \Sigma & 0 & 1 \\
q_0 & \{q_0, q_1\} & \{q_1\} \\
q_1 & \{q_0\} & \emptyset
\end{array}
\]
on the input \( y = 01101 \), we get

\[
\begin{align*}
R_0 &= \{q_0\} \\
R_1 &= \{q_0, q_1\} \\
R_2 &= \{q_1\} \\
R_3 &= \{q_0\} \\
R_4 &= \{q_0, q_1\} \\
R_5 &= \{q_1\}
\end{align*}
\]

And because \( R_5 \cap F = \emptyset \), this NFA does not accept this input.

Formally, the \( R_i \)’s sets is precisely how we convert this NFA into a DFA. This is because the \( R_i \)’s are all members of \( P(Q) \), the power set of \( Q \). That is, the new DFA, \( (Q’, \Sigma, \delta’, q_0’, F’) \) has

\[
Q’ = P(Q)
\]

and transition given by

\[
\delta’(R, a) = \{ \delta(r, a) | r \in R \}.
\]

for each \( R \in Q’ \) and \( a \in \Sigma \). This needs to be modified slightly to take \( \epsilon \) transitions into account. Specifically, for each \( R \subseteq Q \), we define \( E(R) \) as the set of states reachable from \( R \) by traversing 0 or more \( \epsilon \) transitions. Then the transition that take these into account is:

\[
\delta’(R, a) = \bigcup_{r \in R} E(\delta(r, a)).
\]

Finally the new starting state is

\[
q_0’ = E(\{q_0\}),
\]

while the accepting state is all \( R \in P(Q) \) with non-empty intersection with \( F \):

\[
F’ = \{ R \in Q’ | R \cap F \neq \emptyset \}.
\]

Going with the example above, we get a DFA with states

\[
\emptyset, \{q_0\}, \{q_1\}, \{q_0, q_1\}.
\]
Its starting state is \( \{ q_0 \} \), and its accepting states are \( \{ \{ q_1 \}, \{ q_0, q_1 \} \} \). Finally, we can work out its transitions using Equation 1.

\[
\begin{array}{c|cc}
P(Q) \setminus \Sigma & 0 & 1 \\
\emptyset & \emptyset & \emptyset \\
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_1 \} \\
\{ q_1 \} & \{ q_0 \} & \emptyset \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_1 \}
\end{array}
\]