Theory of Languages and Automata

Chapter 1 - Regular Languages & Finite State Automaton

Sharif University of Technology
Finite State Automaton

- We begin with the simplest model of Computation, called \textit{finite state machine} or \textit{finite automaton}.
- They are good models for computers with an extremely limited amount of memory.
  
  → Embedded Systems

- \textit{Markov Chains} are the probabilistic counterpart of Finite Automata.
Simple Example

- Automatic door
Simple Example (cont.)

- State Diagram

- State Transition Table

<table>
<thead>
<tr>
<th></th>
<th>Neither</th>
<th>Front</th>
<th>Rear</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>Open</td>
</tr>
</tbody>
</table>
Formal Definition

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where

1. \(Q\) is a finite set called states,
2. \(\Sigma\) is a finite set called the alphabet,
3. \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function,
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.
Example

$M_1 = (Q, \Sigma, \delta, q_0, F)$, where

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

<table>
<thead>
<tr>
<th></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>

1. $q_1$ is the start state, and
2. $F = \{q_2\}$. 

The finite automaton $M_1$
Language of a Finite machine

If A is the set of all strings that machine M accepts, we say that A is the language of machine M and write: \( L(M) = A \).

We say that M recognizes A or that M accepts A.
Example

$O \ L(M_1) = \{ w \mid w \text{ contains at least one } 1 \text{ and even number of 0s follow the last } 1 \}$. 

The finite automaton $M_1$
Example

- $M_4$ accepts all strings that start and end with a or with b.
Formal Definition

\[ M = (Q, \Sigma, \delta, q_0, F) \]

\[ w = w_1 w_2 \ldots w_n \quad \forall i, w_i \in \Sigma \]

\[ M \text{ accepts } w \iff \exists r_0, r_1, \ldots, r_n \quad \forall i, r_i \in Q \]

1. \[ r_0 = q_0 \]
2. \[ \delta(r_i, w_{i+1}) = r_{i+1}, \text{ for } i = 0, \ldots, n-1, \]
3. \[ r_n \in F. \]
Regular Language

- A language is called a *regular language* if some finite automaton recognizes it.
Example

\[ L(M_5) = \{ w \mid \text{the sum of the symbols in } w \text{ is } 0 \mod 3, \text{ except that } \langle \text{RESET} \rangle \text{ resets the count to } 0 \} \]

✓ As \( M_5 \) recognizes this language, it is a regular language.
Designing Finite Automata

- Put *yourself* in the place of the machine and then see how you would go about performing the machine’s task.

- Design a finite automaton to recognize the regular language of all strings that contain the string 001 as a substring.
There are four possibilities: You
1. haven’t just seen any symbols of the pattern,
2. have just seen a 0,
3. have just seen 00, or
4. have seen the entire pattern 001.
The Regular Operations

- Let $A$ and $B$ be languages. We define the regular operations *union*, *concatenation*, and *star* as follows.

  - **Union**: $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$.

  - **Concatenation**: $A \circ B = \{ xy \mid x \in A \text{ and } y \in B \}$.

  - **Star**: $A^* = \{ x_1x_2\ldots x_k \mid k \geq 0 \text{ and each } x_i \in A \}$. 
Closure Under Union

THEOREM

The class of regular languages is closed under the union operation.
Proof

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognize $A_1$, and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = Q_1 \times Q_2$
2. $\Sigma = \Sigma_1 \cup \Sigma_2$
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.
4. $q_0$ is the pair $(q_1, q_2)$.
5. $F$ is the set of pair in which either members in an accept state of $M_1$ or $M_2$.

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$

$F \neq F_1 \times F_2$
Closure under Concatenation

- **THEOREM**
  - The class of regular languages is closed under the concatenation operation.

- To prove this theorem we introduce a new technique called nondeterminism.
Nondeterminism

O In a *nondeterministic* machine, several choices may exit for the next state at any point.

O Nondeterminism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.
Differences between DFA & NFA

- First, every state of a DFA always has exactly one exiting transition arrow for each symbol in the alphabet. In an NFA, a state may have zero, one, or more exiting arrows for each alphabet symbol.

- Second, in a DFA, labels on the transition arrows are symbols from the alphabet. An NFA may have arrows labeled with members of the alphabet or \( \varepsilon \). Zero, one, or many arrows may exit from each state with the label \( \varepsilon \).
Deterministic vs. Nondeterministic Computation

Deterministic computation
- start
- ...
- accept or reject

Nondeterministic computation
- reject
- ...
- accept
Example

Consider the computation of $N_1$ on input 010110.
Example (cont.)
A **nondeterministic finite automaton** is 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. $\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.
Example

0. $N_1 = (Q, \Sigma, \delta, q_0, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,

2. $\Sigma = \{0,1\}$,

3. $\delta$ is given as

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

1. $q_1$ is the start state, and

2. $F = \{q_4\}$. 

The nondeterministic finite automaton $N_1$ 

![Diagram](image.png)
THEOREM

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

PROOF IDEA convert the NFA into an equivalent DFA that simulates the NFA.

If \( k \) is the number of states of the NFA, so the DFA simulating the NFA will have \( 2^k \) states.
Proof

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing $A$. We construct a DFA $M = (Q', \Sigma', \delta', q'_0, F')$ recognizing $A$.

Let's first consider the easier case wherein $N$ has no $\epsilon$ arrows.

1. $Q' = P(Q)$.
2. $\delta'(R, a) = \bigcup \delta(r, a)$.
3. $q'_0 = \{q_0\}^r_{r \in R}$.
4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$. 

Prof. Movaghar

Theory of Languages and Automata

27
Proof (cont.)

Now we need to consider the $\varepsilon$ arrows.

For $R \subseteq Q$ let

$E(R) = \{q \mid q$ can be reached from $R$ by traveling along 0 or more $\varepsilon$ arrows$\}$.

1. $Q' = P(Q)$.
2. $\delta'(R,a) = \{q \in Q \mid q \in E(\delta(r,a))$ for some $r \in R$\}$.
3. $q_0' = E(\{q_0\})$.
4. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$.
Corollary

A language is regular if and only if some nondeterministic finite automaton recognizes it.
Example

- D’s state set is
  \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.
- The start state is \(E(\{1\}) = \{1,3\}\).
- The accept states are
  \{\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}.

\(N_4 = (Q, \{a, b\}, \delta, 1, \{1\})\)

A DFA \(D\) that is equivalent to the NFA \(N_4\)
Example (cont.)

A DFA $D$ that is equivalent to the NFA $N_4$.

After removing unnecessary states.
CLOSURE UNDER THE REGULAR OPERATIONS [Using NFA]
Closure Under Union

The class of regular languages is closed under the Union operation.

Let NFA1 recognize A1 and NFA2 recognize A2. Construct NFA3 to recognize A1 U A2.
Proof (cont.)

\[ Q = \{q_0\} \cup Q_1 \cup Q_2. \]
The state \( q_0 \) is the start state of \( N \).
The accept states \( F = F_1 \cup F_2. \)
Define \( \delta \) so that for any \( q \in Q \) and any \( a \in \Sigma_\varepsilon \)
\[
\delta(q, a) = \begin{cases}
\delta_1(q, a) & q \in Q_1 \\
\delta_2(q, a) & q \in Q_2 \\
\{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\
\emptyset & q = q_0 \text{ and } a \neq \varepsilon.
\end{cases}
\]
Closure Under Concatenation Operation

The class of regular languages is closed under the concatenation operation.
Proof (cont.)

\[ Q = Q_1 \cup Q_2. \]

The states of \( N \) are all the states of \( N_1 \) and \( N_2 \).

The state \( q_1 \) is the same as the start state of \( N_1 \).

The accept states \( F_2 \) are the same as the accept states of \( N_2 \).

Define \( \delta \) so that for any \( q \in Q \) and any \( a \in \Sigma \),

\[
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\
\delta_2(q, a) & q \in Q_2.
\end{cases}
\]
Closure Under Star operation

- The class of regular languages is closed under the star operation.

- We represent another NFA to recognize $A^*$. 
Proof (cont.)

1. \( Q = \{q_0\} \cup Q_1 \) The states of \( N \) are the states of \( N_1 \) plus a new start state.
2. The state \( q_0 \) is the new start state.
3. \( F = \{q_0\} \cup F_1 \)
4. The accept states are the old accept states plus the new start state.
5. Define \( \delta \) so that for any \( q \in Q \) and \( a \in \Sigma \):

\[
\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\
\{q_1\} & q = q_0 \text{ and } a = \varepsilon \\
\emptyset & q = q_0 \text{ and } a \neq \varepsilon.
\end{cases}
\]
Say that $R$ is a regular expression if $R$ is:

1. $a$ for some $a$ in the alphabet $\Sigma$
2. $\varepsilon$
3. $\emptyset$
4. $(R_1 U R_2)$, where $R_1$ and $R_2$ are regular exp.
5. $(R_1 o R_2)$, where $R_1$ and $R_2$ are regular exp.
6. $(R_1^*)$, where $R_1$ and $R_2$ are regular exp.

Recursive Definition?
Let $R$ be a regular expression. $L(R)$ is the language that is defined by $R$:
1. if $R = a$ for $a \in \Sigma$ then $L(R) = \{a\}$
2. if $R = \varepsilon$ then $L(R) = \{\varepsilon\}$
3. if $R = \emptyset$ then $L(R) = \emptyset$
4. if $R = R_1 U R_2$ then $L(R) = L(R_1) U L(R_2)$
5. if $R = R_1 \circ R_2$ then $L(R) = L(R_1) \circ L(R_2)$
6. if $R = R_1^*$ then $L(R) = (L(R_1))^*$
Examples (cont.)

1. \[(0 \cup \varepsilon)1^* = 01^* \cup 1^*\]
2. \[\Sigma^*1\Sigma^* = \{w|w \text{ contains at least one } 1\}\]
3. \[0^*10^* = \{w|w \text{ contains a single } 1\}\]
4. \[\Sigma^*001\Sigma^* = \{w|w \text{ contains } 001 \text{ as a substring}\}\]
5. \[01U10 = \{01,10\}\]
6. \[(\Sigma\Sigma)^* = \{w|w \text{ is a string of even length}\}\]
7. \[(\Sigma\Sigma\Sigma)^* = \{w|\text{the length of } w \text{ is a multiple of } 3\}\]
8. \[(\Sigma\Sigma\Sigma\Sigma)^* = \{w|\text{the length of } w \text{ is a multiple of } 4\}\]
9. \[(0 \cup \varepsilon)1^* = 01^* \cup 1^*\]
10. \[(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0,1,01\}\]
11. \[1^*\emptyset = \emptyset\]
12. \[\emptyset^* = \{\varepsilon\}\]
Equivalence of DFA and Regular Expression

- A language is regular if and only if some regular expression describes it.

**Lemma:**
- If a language is described by a regular expression, then it is regular.
- If a language is regular, then it is described by a regular expression.
We consider the six cases in the formal definition of regular expressions:

1. $R = a$ for some $a$ in $\Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.

2. $R = \epsilon$. Then $L(R) = \{\epsilon\}$, and the following NFA recognizes $L(R)$.

3. $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes $L(R)$.

4. $R = R_1 \cup R_2$.

5. $R = R_1 \circ R_2$.

6. $R = R_1^*$.
Examples

Building an NFA from the regular expression \((ab \cup a)^*\)
We need to show that, if a language $A$ is regular, a regular expression describes it!

First we show how to convert DFAs into GNFAs, and then GNFAs into regular expressions.

We can easily convert a DFA into a GNFA in the special form.
A generalized nondeterministic finite automaton is a 5-tuple, a 5-tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\), where

1. \(Q\) is a finite set called \textit{states},
2. \(\Sigma\) is a the input \textit{alphabet},
3. \(\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{Q_{\text{start}}\} \rightarrow R\) is the \textit{transition function},
4. \(q_{\text{start}}\) is the \textit{start state}, and
5. \(q_{\text{accept}}\) is the \textit{accept state}. 

**Formal Definition**

Theory of Languages and Automata

Prof. Movaghar
Assumptions

For convenience we require that GNFAs always have a special form that meets the following conditions:

1. The start state has transition arrows going to every other state but no arrows coming in from any other state.
2. There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
3. Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.
Acceptance of Languages for GNFA

A GNFA accepts a string $w$ in $\Sigma^*$ if $w = w_1 w_2 \ldots w_k$, where each $w_i$ is in $\Sigma^*$ is in $\Sigma^*$ and a sequence of $q_0, q_1, ..., q_k$ exists such that

1. $q_0 = q_{\text{start}}$ is the start state,
2. $q_k = q_{\text{accept}}$ is the accept state, and
3. For each $i$, we have $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$; in other words $R_i$ is the expression on the arrow from $q_{i-1}$ to $q_i$. 
How to Eliminate a State?
Example

Converting a two-state DFA to an equivalent regular expression
Example

(a)  

(b)  

(c)  

(d)  

(e)  

\[(a(aa \cup b) * ab \cup b)((ba \cup a)(aa \cup b) * ab \cup bb)*((ba \cup a)(aa \cup b)^* \cup \varepsilon) \cup a(aa \cup b)^*\]
Grammar

A grammar $G$ is a 4-tuple $G = (V, \Sigma, R, S)$ where:

1. $V$ is a finite set of **variables**, 
2. $\Sigma$ is a finite, disjoint from $V$, of **terminals**, 
3. $R$ is a finite set of **rules**, 
4. $S$ is the **start** variable.
A rule is of the form

\[ x \rightarrow y \]

where \( x \in (V \cup \Sigma)^+ \) and \( y \in (V \cup \Sigma)^* \)

The rules are applied in the following manner: given a string \( w \) of the form

\[ w = uxv, \]

We say that the rule \( x \rightarrow y \) is applicable to this string, and we may use it to replace \( x \) with \( y \), thereby obtaining a new string

\[ z = uyv, \]

This is written as

\[ w \Rightarrow z. \]
Derivation

If

\[ W_1 \Rightarrow W_2 \Rightarrow \ldots \Rightarrow W_n \]

we say that \( W_1 \) derives \( W_n \) and write

\[ W_1 \Rightarrow^* W_n \]

Thus, we always have

\[ W \Rightarrow^* W \]
Language of a Grammar

Let $G = (V, \Sigma, R, S)$ be a grammar. Then, the set

$$L(G) = \{W \in \Sigma^*: S \Rightarrow^* W\}$$

is the *language generated* by $G$. 

Example

Consider the grammar

\[ G = \{ \{S\}, \{a,b\}, P, S\} \]

with \( P \) given by

\[ S \rightarrow aSb \]
\[ S \rightarrow \varepsilon \]

Then

\[ S \rightarrow aSb \rightarrow aaSbb \rightarrow aabb \]

So we can write

\[ S \Rightarrow^* aabb \]

Then,

\[ L(G) = \{a^n b^n: n \geq 0\} \]
A Notation for Grammars

Consider the grammar

\[ G = (\{S\}, \{a,b\}, P, S) \]

with \( P \) given by

\[ S \rightarrow aSb \]
\[ S \rightarrow \varepsilon \]

The above grammar is usually written as:

\[ G: S \rightarrow aSb | \varepsilon \]
Regular Grammar

A grammar $G = (V, \Sigma, R, S)$ is said to be **right-linear** if all rules are of the form

$$A \rightarrow xB$$

$$A \rightarrow x$$

Where $A, B \in V$, and $X \in \Sigma^*$. A grammar is said to be **left-linear** if all rules are of the form

$$A \rightarrow Bx$$

$$A \rightarrow x$$

A regular grammar is one that is either right-linear or left-linear.
Let $G = (V, \Sigma, R, S)$ be a right-linear grammar. Then:

L(G) is a regular language.
Example

Construct a NFA that accepts the language generated by the grammar

\[ V_0 \to aV_1 \]
\[ V_1 \to abV_0 \mid b \]
Theorem
Let $L$ be a regular language on the alphabet $\Sigma$.
Then:
There exists a right-linear grammar $G = (V, \Sigma, R, S)$
Such that $L = L(G)$. 
Theorem

A language is regular if and only if there exists a left-linear grammar $G$ such that $L = L(G)$.

Outline of the proof:
Given any left-linear grammar with rules of the form

- $A \rightarrow Bx$
- $A \rightarrow x$

We can construct a right-linear $\hat{G}$ by replacing every such rule of $G$ with

- $A \rightarrow x^RB$
- $A \rightarrow x^R$

We have $L(G) = L(\hat{G})^R$. 
Theorem

A language $L$ is regular *if and only if* there exists a regular grammar $G$ such that $L = L(G)$.

$L$ is *regular* $\iff \exists \ G: L = L(G)\mid G$ is *regular*