Let me show you a machine so simple that you can understand it in less than two minutes.

The machine accepts a string if the process ends in a double circle.

Anatomy of a Deterministic Finite Automaton

The alphabet of a finite automaton is the set where the symbols come from: \( \{0, 1\} \)
The language of a finite automaton is the set of strings that it accepts.
The Language of Machine $M$

$L(M) = \{\text{all strings of 0s and 1s}\}$

$L(M) = \{w \mid w \text{ has an even number of 1s}\}$

**Notation**

An alphabet $\Sigma$ is a finite set (e.g., $\Sigma = \{0, 1\}$)

A string over $\Sigma$ is a finite-length sequence of elements of $\Sigma$. The set of all strings over $\Sigma$ is denoted by $\Sigma^*$.

For $x$ a string, $|x|$ is the length of $x$.

The unique string of length 0 will be denoted by $\epsilon$ and will be called the empty or null string.

A language over $\Sigma$ is a set of strings over $\Sigma$,

$L(\text{Language}) \in \Sigma^*$

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

$Q$ is the set of states

$\Sigma$ is the alphabet

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept states

$L(M) = \text{the language of machine } M$

$= \text{set of all strings machine } M \text{ accepts}$

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

$Q = \{q_0, q_1, q_2, q_3\}$

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

$q_0 \in Q$ is start state

$F = \{q_1, q_2\} \subseteq Q$ accept states

$\delta : Q \times \Sigma \rightarrow Q$ transition function

<table>
<thead>
<tr>
<th>$\delta$</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>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

“ABA” The Automaton

<table>
<thead>
<tr>
<th>Input String</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>aba</td>
<td>Accept</td>
</tr>
<tr>
<td>aabb</td>
<td>Reject</td>
</tr>
<tr>
<td>aabba</td>
<td>Accept</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Accept</td>
</tr>
</tbody>
</table>
What machine accepts this language?

$L = $ all strings in $\{a,b\}^*$ that contain at least one $a$

What machine accepts this language?

$L =$ strings with an odd number of $b$'s and any number of $a$'s

What is the language accepted by this machine?

$L =$ any string ending with a $b$

What is the language accepted by this machine?

$L(M) =$ any string with at least two $a$'s

What machine accepts this language?

$L =$ any string with an $a$ and a $b$

What machine accepts this language?

$L =$ strings with an even number of $ab$ pairs
The “Grep” Problem
Input: Text T of length t, string S of length n
Problem: Does string S appear inside text T?
Naive method:
\[ a_1, a_2, a_3, a_4, a_5, \ldots, a_t \]
Cost: Roughly nt comparisons

Automata Solution
Build a machine M that accepts any string with S as a consecutive substring
Feed the text to M
Cost: t comparisons + time to build M
As luck would have it, the Knuth, Morris, Pratt algorithm builds M quickly

Real-life Uses of DFAs
Grep
Coke Machines
Thermostats (fridge)
Elevators
Train Track Switches
Lexical Analyzers for Parsers

A language is regular if it is recognized by a deterministic finite automaton
\[ L = \{ w \mid w \text{ contains } 001 \} \text{ is regular} \]
\[ L = \{ w \mid w \text{ has an even number of } 1s \} \text{ is regular} \]
**Union Theorem**

Given two languages, \( L_1 \) and \( L_2 \), define the union of \( L_1 \) and \( L_2 \) as

\[
L_1 \cup L_2 = \{ w \mid w \in L_1 \text{ or } w \in L_2 \}
\]

Theorem: The union of two regular languages is also a regular language

Proof Sketch: Let

\[ M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1) \] be finite automaton for \( L_1 \)

and

\[ M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2) \] be finite automaton for \( L_2 \)

We want to construct a finite automaton

\[ M = (Q, \Sigma, \delta, q_0, F) \] that recognizes \( L = L_1 \cup L_2 \)

**Automaton for Union**

\[
\begin{array}{c}
\text{q}_0, p_0 \quad \quad 1 \quad \quad \text{q}_1, p_0 \\
0 \quad 0 \\
\text{q}_0, p_1 \quad \quad 1 \quad \quad \text{q}_1, p_1
\end{array}
\]

**Theorem: The union of two regular languages is also a regular language**

**Automaton for Intersection**

\[
\begin{array}{c}
\text{q}_0, p_0 \quad \quad 1 \quad \quad \text{q}_1, p_0 \\
0 \quad 0 \\
\text{q}_0, p_1 \quad \quad 1 \quad \quad \text{q}_1, p_1
\end{array}
\]
Theorem: The union of two regular languages is also a regular language

Corollary: Any finite language is regular

The Regular Operations

Union: \( A \cup B = \{ w \mid w \in A \text{ or } w \in B \} \)

Intersection: \( A \cap B = \{ w \mid w \in A \text{ and } w \in B \} \)

Reverse: \( A^R = \{ w_k \ldots w_1 \mid w_k \ldots w_1 \in A \} \)

Negation: \( \neg A = \{ w \mid w \not\in A \} \)

Concatenation: \( A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \} \)

Star: \( A^* = \{ w_k \ldots w_1 \mid k \geq 0 \text{ and each } w_i \in A \} \)

Regular Languages Are Closed Under The Regular Operations

We have seen part of the proof for Union. The proof for intersection is very similar. The proof for negation is easy.

Regular Languages Are Not Regular

Consider the language \( L = \{ a^n b^n \mid n > 0 \} \)

i.e., a bunch of a’s followed by an equal number of b’s

No finite automaton accepts this language

Can you prove this?

\( a^n b^n \) is not regular.

No machine has enough states to keep track of the number of a’s it might encounter
That is a fairly weak argument
Consider the following example...

$L = \text{strings where the \# of occurrences of the pattern } ab \text{ is equal to the number of occurrences of the pattern } ba$

Can't be regular. No machine has enough states to keep track of the number of occurrences of ab

M accepts only the strings with an equal number of ab’s and ba’s!

Let me show you a professional strength proof that $a^n b^n$ is not regular...

Pigeonhole principle:
Given n boxes and m > n objects, at least one box must contain more than one object

Letterbox principle:
If the average number of letters per box is x, then some box will have at least x letters (similarly, some box has at most x)

Theorem: $L = \{a^n b^n \ | \ n > 0 \}$ is not regular
Proof (by contradiction):
Assume that L is regular
Then there exists a machine M with k states that accepts L
For each $0 \leq i \leq k$, let $S_i$ be the state M is in after reading $a^i$
$\exists i, j \leq k$ such that $S_i = S_j$, but $i \neq j$
M will do the same thing on $a^i b^i$ and $a^j b^i$
But a valid M must reject $a^i b^i$ and accept $a^j b^i$
You can learn much more about these creatures in the FLAC course.

Formal Languages, Automata, and Computation

• There is a unique smallest automaton for any regular language
• It can be found by a fast algorithm.

Deterministic Finite Automata
• Definition
• Testing if they accept a string
• Building automata

Regular Languages
• Definition
• Closed Under Union, Intersection, Negation
• Using Pigeonhole Principle to show language ain’t regular

Here’s What You Need to Know…