One Minute To Learn Programming: Finite Automata
Today we’ll talk about a programming language so simple that you can learn it in less than a minute.
Meet "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>ε</td>
<td>Accept</td>
</tr>
</tbody>
</table>
The Simplest Interesting Machine:

Finite State Machine

OR

Finite Automaton
## Finite Automaton

<table>
<thead>
<tr>
<th>Friendly</th>
<th>Formal, “unfriendly”</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Finite set of states</th>
<th>$Q = {q_o, q_1, q_2, \ldots, q_k}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A start state</td>
<td>$q_o$</td>
</tr>
<tr>
<td>A set of accepting states</td>
<td>$F = {q_{i_1}, q_{i_2}, \ldots, q_{i_r}}$</td>
</tr>
<tr>
<td>A finite alphabet</td>
<td>$\sum$</td>
</tr>
<tr>
<td>State transition instructions</td>
<td>$\partial : Q \times \Sigma \to Q$</td>
</tr>
</tbody>
</table>

- $\partial(q_i, a) = q_j$
How Machine $M$ operates.

$M$ "reads" one letter at a time from the input string (going from left to right)

$M$ starts in state $q_0$.
If $M$ is in state $q_i$ reads the letter $a$ then

If $\delta(q_i, a)$ is undefined then CRASH.

Otherwise $M$ moves to state $\delta(q_i, a)$
Let $M = (Q, \Sigma, F, \delta)$ be a finite automaton.

$M$ **accepts** the string $x$ if when $M$ reads $x$ it ends in an accepting state.

$M$ **rejects** the string $x$ if when $M$ reads $x$ it ends in a non-accepting state.

$M$ **crashes** on $x$ if $M$ crashes while reading $x$. 
The set (or language) accepted by $M$ is:

$$L_M = \{ x \in \Sigma^* \mid M \text{ accepts } x \}$$

$$\Sigma^k \equiv \text{All length } k \text{ strings over the alphabet } \Sigma$$

$$\Sigma^* \equiv \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

Notice that this is $\{ \varepsilon \}$.
Back to “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>ε</td>
<td>Accept</td>
</tr>
</tbody>
</table>
What is the language accepted by this machine?

$L = \{a, b\}^* = \text{all finite strings of } a\text{'s and } b\text{'s}$
What is the language accepted by this machine?

\[ a, b \]
What is the language accepted by this machine?

$L = \text{all even length strings of } a \text{'s and } b \text{'s}$
What machine accepts this language?

\[ L = \text{all strings in} \ \{a,b\}^* \text{ that contain at least one} \ a \]
What machine accepts this language?

$L = \text{strings with an odd number of } b\text{'s and any number of } a\text{'s}$
What is the language accepted by this machine?

$L = $ any string ending with $a$ followed by $b$
What is the language accepted by this machine?
What is the language accepted by this machine?

$L = \text{any string with at least two } a\text{'s}$
What machine accepts this language?

$L = \text{any string with an } a \text{ and a } b$
What machine accepts this language?

$L = \text{any string with an } a \text{ and a } b$
What machine accepts this language?

$L = \text{strings with an even number of } ab \text{ pairs}$
\[ L = \text{all strings containing } ababb \text{ as a consecutive substring} \]
\( L = \) all strings containing \textit{ababb} as a consecutive substring

Invariant: I am state \textit{s} exactly when \textit{s} is the longest suffix of the input (so far) that forms a prefix of \textit{ababb}.
The “grep” Problem

Input:
- text \( T \) of length \( t \)
- string \( S \) of length \( n \)

Problem:
Does the string \( S \) appear inside the text \( T \)?

Naïve method:

Cost: \( O(nt) \) 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 finite state machines

• grep
• coke machines
• thermostats (fridge)
• elevators
• train track switches
• lexical analyzers for parsers
Any $L \subseteq \Sigma^*$ is defined to be a language.

$L$ is just a set of strings. It is called a language for historical reasons.
Let $L$ be a language.

$L$ is called a regular language if there is some finite automaton that accepts $L$.

In this lecture we have seen many regular languages.

- even length strings
- strings containing $ababb$
Theorem: Any finite language is regular.

Proof: Make a machine with a "path" for each string in the language, sharing prefixes.

Example: \( L = \{a, bcd, ac, bb\} \)
Are all languages regular?
Consider the language

\[ a^n b^n = \{ \varepsilon, ab, aabb, aaabbb, \ldots \} \]

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...
$L =$ strings where the # of occurrences of the pattern $ab$ 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 $ab$. 
Remember “ABA”?

ABA 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 $a$, then some box will have at least $a$ letters. (Similarly, some box has at most $a$.)
Theorem: $a^n b^n$ is not regular.

Proof: Assume that it is. Then $\exists \ M$ with $k$ states that accepts it.

For each $0 \leq i \leq k$, let $S_i$ be the state $M$ is in after reading $a^i$.

$\exists i, j \leq k$ s.t. $S_i = S_j$, 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 $a^i b^j$.
MORAL:

Finite automata can't count.
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.