Finite Automata

1 September 2022
What problems can we solve with a computer?
What problems can we solve with a computer?

What kind of computer?
Two challenges

Computers are dramatically better now than they’ve ever been, and we expect that trend to continue!

Writing proofs on formal definitions is hard, and today’s computers are already way more complicated than the mathematical structures you wrote proofs about in CMPU 145.
How can we prove what computers can and can’t do…
…so our results are still true in 20 years?
…without multi-hundred-page proofs?
Enter automata

An *automaton* is a mathematical model of a computing device.

It’s an abstraction of a real computer, like how graphs are abstractions of social networks, transportation grids, etc.
The automata we’ll explore are

Powerful enough to capture huge classes of computer devices, but
Simple enough that we can reason about them in a small space!
What do these automata look like?
A tale of two computers
A tale of two computers
A tale of two computers
<table>
<thead>
<tr>
<th><strong>Basic Calculator</strong></th>
<th><strong>Laptop</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Small amount of memory</td>
<td>Large amount of memory</td>
</tr>
<tr>
<td>Fixed set of functions</td>
<td>Reprogrammable; run lots</td>
</tr>
<tr>
<td></td>
<td>of different programs</td>
</tr>
</tbody>
</table>
**Basic Calculator**

- Small amount of memory
- Fixed set of functions

**Laptop**

- Large amount of memory
- Reprogrammable; run lots of different programs
Computing with finite memory
Data stored electronically
Algorithm is in silicon
Memory limited by display
Data stored electronically

Algorithm is in silicon

Memory limited by display
Data stored electronically
Algorithm is in silicon
Memory limited by display

Data stored in wood
Algorithm is in your brain
Memory limited by beads
How do we model “memory” and “an algorithm” when they can take on so many forms?
Focus on what’s in common:

The machines *receive input* from an external source.

That input is provided *sequentially*, one discrete unit at a time.

Each input causes the device to *change configuration*. This change, big or small, is where the computation happens!

Once all input is provided, we can *read off an answer* based on the configuration of the device.
Finite automata
A *finite automaton* – also called a *finite-state machine* – is the simplest model of a computer that’s still interesting to study.
Each finite automaton consists of a set of states connected by transitions.
Each circle represents a state of the automaton.
One state is designated as the start state, indicated by an arrow
The automaton is run on an input string.
The automaton can be in one state at a time. It begins in the start state.
The automaton now begins processing characters in the order in which they appear.
Each arrow in this diagram represents a transition. The automaton always follows the transition for the symbol being read.
A finite automaton with the following states and transitions:

- **States:** q0, q1, q2, q3
- **Start State:** q0
- **Transition Labels:**
  - From q0 to q1 on input 0
  - From q0 to q3 on input 1
  - From q1 to q0 on input 0
  - From q1 to q2 on input 1
  - From q2 to q1 on input 0
  - From q3 to q0 on input 1

The automaton transitions as follows:

- Start at q0.
- If the input is 0, move to q1.
- If the input is 1, move to q3.
- If the input is 0 while at q1, move back to q0.
- If the input is 1 while at q1, move to q2.
- If the input is 0 while at q2, move back to q1.
- If the input is 1 while at q3, move to q0.

The diagram also shows the binary sequence 010110 as a path through the automaton starting from q0.
After transitioning, the automaton considers the next symbol in the input.
Now that the automaton has read all of the input, it can decide whether to accept or reject
Now that the automaton has read all of the input, it can decide whether to accept or reject.

The double circle indicates this is an **accept state**, so it accepts!
Illustration by
Gemma Correll
Let’s try another input

![State Machine Diagram]

1 0 1
Let’s try another input
This state is not an accept state, so the automaton rejects.
This state is not an accept state, so the automaton rejects.

Illustration by Gemma Correll
A finite automaton consists of a set of states connected by transitions.

One state is designated the start state.

Some states are accept states.

Transition arcs are labeled with one or more symbols from some alphabet.
An automaton processes a string by beginning in the start state and following the indicated transitions.

The new state is completely determined by the current state and the symbol it just read.

When the input is exhausted,

If the automaton is in an accepting state, it accepts the input.
Otherwise, it rejects the input.
A finite automaton does not accept as soon as it enters an accept state.

It only accepts if it ends in an accept state.
Finite-state machines are all around us.
Finite automaton for a newspaper vending machine
Other real-world finite automata?
Finite automata in action

Used in

- Text editors and search engines for pattern matching
- Compilers for lexical analysis
- Web browsers for HTML parsing

Serve as the control unit in many physical systems, including

- Elevators, traffic signals, vending machines
- Computer microprocessors
- Network protocol stacks and old VCR clocks

Play a key role in natural language processing and machine learning

*Markov chains* are probabilistic FAs used in part of speech tagging, speech processing, and optical character recognition.
Formal language theory
What problems can we solve with a computer?
What problems can we solve with a computer?

What’s a “problem”?
Before we can talk about what problems we can solve, we need a formal definition of a “problem”.

We want a definition that

- corresponds to the problems we want to solve,
- captures a large class of problems, and
- is mathematically simple to reason about
Virtually all computational problems can be recast as *language recognition problems*.

E.g., the problem of determining whether an integer is prime:

*Problem*: Is 97 prime?

*Recast*: Is the string 97 in the language of all primes, \{2, 3, 5, 7, 13, \ldots\}?
Strings and languages
An *alphabet*, denoted $\Sigma$, is a finite, non-empty set of symbols called *characters*, e.g.,

**Binary:** $\Sigma = \{0, 1\}$

**ASCII:** $\Sigma = \{a, b, c, \ldots, 0, 1, \ldots, !, @, #, \ldots\}$
A *string* over an alphabet $\Sigma$ is a finite sequence of characters drawn from $\Sigma$.

For example, if $\Sigma = \{a, b\}$, then

- $a$

and

- $abbaba$

are strings over $\Sigma$.

The *empty string*, denoted $\varepsilon$, has no characters.
A **string** over an alphabet \( \Sigma \) is a **finite** sequence of characters drawn from \( \Sigma \).

For example, if \( \Sigma = \{a, b\} \), then

- \( a \)
- and
- \( abbaba \)

are strings over \( \Sigma \).

The **empty string**, denoted \( \varepsilon \), has no characters.
A formal *language* is a set of strings, e.g.,

∅, the *empty set*, is a language of zero strings

{kitty, cat} is a language of two strings

We say that $L$ is a *language over* $\Sigma$ if it is a set of strings over $\Sigma$, e.g.,

The language of palindromes over $\Sigma = \{a, b, c\}$ is the infinite set

{ε, a, b, c, aa, bb, cc, aaa, aba, aca, bab, …}
The set of all strings composed from letters in $\Sigma$ is denoted $\Sigma^*$. So, formally, we say that $L$ is a language over $\Sigma$ if $L \subseteq \Sigma^*$. 
Mathematical lookalikes

We now have $\epsilon$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😞
Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😞

$\in$ is the element-of relation.
Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😞

$\in$ is the element-of relation.

$\varepsilon$ is the empty string.
Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😞

$\in$ is the element-of relation.

$\varepsilon$ is the empty string.

$\Sigma$ is an alphabet.
Mathematical lookalikes

We now have $\in$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😞

$\in$ is the element-of relation.
$\varepsilon$ is the empty string.
$\Sigma$ is an alphabet.
$\Sigma^*$ means “all strings that can be made from characters in $\Sigma$”.
Mathematical lookalikes

We now have $\varepsilon$, $\varepsilon$, $\Sigma$, and $\Sigma^*$ 😞

$\varepsilon$ is the element-of relation.
$\varepsilon$ is the empty string.
$\Sigma$ is an alphabet.
$\Sigma^*$ means “all strings that can be made from characters in $\Sigma$”.

This means we can write things like

We have $\varepsilon \in \Sigma^*$, but $\varepsilon \notin \Sigma$

which is true!
Languages are sets of Strings, which are finite sequences of Characters. Alphabets are finite, nonempty sets of Characters.
The **language of a finite automaton** is the set of strings that it accepts, i.e., strings that label paths that go from the start state to some accepting state.

If $M$ is an automaton that processes characters from the alphabet $\Sigma$, then its language is defined as

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$
The story so far

A finite automaton is a collection of states joined by transitions.

Some state is designated as the start state.

Some number of states are designated as accepting states.

The automaton processes a string by beginning in the start state and following the indicated transitions.

If the automaton ends in an accepting state, it accepts the input.

Otherwise, the automaton rejects the input.

The language of an automaton is the set of strings it accepts.
Exercise

\[ L(M) = \{ ? \} \]
Exercise

$L(M) = \{ w \mid w \in \{0, 1\}^* \text{ and } w \text{ does not have suffix 1}\}$
Exercise

Design a finite automaton to recognize decimal numbers.
Acknowledgments

This lecture incorporates material from:

Phyllis Frankl, New York University
W. Daniel Hillis, *The Pattern on the Stone*
Nancy Ide, Vassar College
Keith Schwarz, Stanford University
Michael Sipser, *Introduction to the Theory of Computation*
Jeffrey Ullman, Stanford University