What good are DFAs?

Draw a DFA that accepts the following language:

$L = \{w \mid w \text{ is } 010 \text{ or the length of } w \text{ is even}\}$

You may want to practice on scratch paper first.

(Your response)
\[ L = \{ w \mid w \text{ is } 010 \text{ or the length of } w \text{ is even} \} \]

Did you write some test cases?

Does it have an initial state and at least one accepting state?

Does every state have a transition for 0 and a transition for 1?

Does it accept the empty string?

Does it accept 0100 and 0101?

Does it have at least 6 states?
What is Computer Science?

Given a problem
Is there a solution?
What is it?
How good is it?

What do we mean by "problem"?
There's not always!
How would we know?!

← programming
(but also algorithms, data structures, ...)

What do we mean by "good"?
How do we measure goodness?
What is Computer Science?

Given a computational problem:
- Is there a solution?
- What is it?
- How good is it?

What do we mean by "computational problem"?
There's not always! How would we know?!

What do we mean by "good"?
How do we measure goodness?

← programming
(but also algorithms, data structures, ...)

CS 42 in one slide
What do we mean by “problem”?

Our working model: a simplification of “problem” that is good enough for now

A computational problem is a question that we can write down and give to a computer, which can give us an answer.

encoded in binary

"yes" or "no" (decision problem)
What do we mean by “computer”?

A computational problem is a question that we can write down and give to a computer, which can give us an answer. encoded in binary

"yes" or "no" (decision problem)
What do we mean by “computer”?

A computational problem is a question that we can write down and give to a computer, which can give us an answer. "yes" or "no" (decision problem)
Is a DFA a good enough model?

A computational problem is a question that we can write down and give to a computer, which can give us an answer. 

"yes" or "no" (decision problem)
All-DNA finite-state automata with finite memory
Zhen-Gang Wang1, Johann Elbaz1, F. Remacle2, R. D. Levine2,3, and Itamar Willner2,3

Biomolecular logic devices can be applied for sensing and nanomedicine. We built three DNA tweezers that are activated by the inputs $\text{H}^+/	ext{OH}^-; \text{Hg}^{2+}/\text{cysteine}$; nucleic acid linker/complementary antilinker to yield a 16-states finite-state automaton. The outputs of the automata are the configuration of the respective tweezers (open or closed) determined by observing fluorescence from a fluorophore/quencher pair at the end of the arms of the tweezers. The system exhibits a memory because each current state and output depend not only on the source configuration but also on past states and inputs.

Fig. 1. (A) General scheme for the application of three-tweezers structures and a set of counteracting inputs $I_1$ and $I_2$ to yield eight different configurations (outputs) in a finite-state automaton. (B) General scheme for the

Contributed by Raphael D. Levine, October 25, 2010 (sent for review August 6, 2010)


The authors declare no conflict of interest.

1Institute of Chemistry, Hebrew University of Jerusalem, Jerusalem 91904, Israel; 2Chemistry Department, B6c, University of Liège, 4000 Liège, Belgium; and 3Department of Chemistry and Biochemistry, Crump Institute for Molecular Imaging, and Department of Molecular and Medical Pharmacology, University of California, Los Angeles, CA 90095

PNAS no. 51 www.pnas.org/cgi/doi/10.1073/pnas.1015858107

xkcd.com/208
DFAs $\equiv$ Regular Expressions

(regular languages)
Regular expressions

shorthand notation to describe a regular language

\[ L = \{ab\} \]
Regular expressions

shorthand notation to describe a regular language

\[ L = \{ w \mid w \text{ starts and ends with } a \text{ and has 0 or more } b \text{ s in the middle} \} \]
Regular expressions
shorthand notation to describe a regular language

\[ L = \{ h, m, c \} \]

```
L = {h, m, c}
```

```
L = \{ h, m, c \}
```

Union of a regular languages
Regular expressions

shorthand notation to describe a regular language

\[ L = \{010, 101\} \]

\[(010) + (101)\]

010
101
Regular expressions

shorthand notation to describe a regular language

$L = \{w \mid w \text{ is } 010 \text{ or the length of } w \text{ is even}\}$

(010) + ( ((0+1)(0+1))^* )

010
λ
00
...
0101
Deterministic Finite Automaton

Formal definition

A machine $M$ that consists of:

- an alphabet $\Sigma$
- a finite set of states, including:
  - initial state
  - accepting state(s)
- transitions between states

for every state, every letter in $\Sigma$ labels one and only one transition

Given a string $w \in \Sigma^*$, $M$ accepts $w$ if consuming $w$ causes $M$ to terminate in an accepting state.
Non-deterministic Finite Automaton

A machine $M$ that consists of:

- an alphabet $\Sigma$
- a finite set of states, including:
  - initial state
  - accepting state(s)
- transitions between states
  - for every state, every letter in $\Sigma$ labels one and only one transition
  - can also be a $\lambda$ transition

Given a string $w \in \Sigma^*$, $M$ accepts $w$ if consuming $w$ causes $M$ to terminate in an accepting state.
Draw these NFAs

(1) \( L = \{ w \mid w \text{ is 010 or the length of } w \text{ is even}\} \)

(2) \( L = \{ w \mid w \text{ starts and ends with } 01\} \)  

(3) \( L = \{ \text{www.x.com} \mid x \in [a-z]^+\} \)

Write test cases first!!!!
At least three strings accepted by the NFA.
At least three strings rejected by the NFA.
(1) \( L = \{ w \mid w \text{ is } 010 \text{ or the length of } w \text{ is even} \} \)

Write test cases first!!!!

At least three strings accepted by the NFA.
At least three strings rejected by the NFA.
(2) \( L = \{ w \mid w \text{ starts and ends with } 01 \} \)

Write test cases first!!!!
At least three strings **accepted** by the NFA.
At least three strings **rejected** by the NFA.
Write test cases first!!!!
At least three strings **accepted** by the NFA.
At least three strings **rejected** by the NFA.
NFAs \equiv DFAs \equiv Regular Expressions (regular languages)
Equivalent modes of computation

Which would you rather use?

(1) NFAs
(2) DFAs
(3) Regular expressions
(4) It depends
The benefits of equivalent models?

(1) Express problem as regular expression (RE)

(2) Convert RE to NFA

(3) Convert NFA to DFA

(4) Minimize DFA

(5) Profit!
Regular Languages FTW!
or: How to Cheat at Boggle
Regular Languages FTW?

No more math homework?

(1) \( L = \{a^N b^N \mid N > 0\} \) // equality

(2) \( L = \{a^N b^{2N} \mid N > 0\} \) // multiplication

(3) \( L = \{a^N b^M c^{(N+M)} \mid N, M > 0\} \) // addition

Not Regular
\( \nexists \) a DFA that accepts \( L \)
Distinguishability theorem

If a set \( S = \{ w_1, w_2, \ldots, w_n \} \) is pairwise distinguishable for a language \( L \), then any FSM that accepts \( L \) must have at least \( n \) states.

\[
L = \{ w \mid w \text{'s length is divisible by 3} \}
\]

L requires at least 3 states.

\[
\begin{array}{c|c|c|c}
 w_1 & w_2 & w_3 \\
\hline
 \lambda & 1 & 11 \\
 \hline
 \lambda & n/a & 11 & 1 \\
 \hline
 1 & \text{redundant} & n/a & 1 \\
 \hline
 11 & \text{redundant} & \text{redundant} & n/a \\
\end{array}
\]
Myhill–Nerode theorem

A language L is **regular** if and only if we can define a set \( S \) of pairwise distinguishable strings such that \( |S| \) is **finite**.

So...how can we prove that L is *not* regular?
Prove that $L = \{a^N b^N \mid N > 0\}$ is not regular.

Let $S = aa^*$

<table>
<thead>
<tr>
<th>$w_i$</th>
<th>$w_j$</th>
<th>$z$</th>
<th>Accept $w_i z$</th>
<th>Reject $w_j z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>aa</td>
<td>b</td>
<td>ab</td>
<td>aab</td>
</tr>
<tr>
<td>a</td>
<td>aaa</td>
<td>b</td>
<td>ab</td>
<td>aaab</td>
</tr>
<tr>
<td>a</td>
<td>aaaa</td>
<td>b</td>
<td>ab</td>
<td>aaaaab</td>
</tr>
<tr>
<td>a</td>
<td>aaaaa</td>
<td>b</td>
<td>ab</td>
<td>aaaaaab</td>
</tr>
</tbody>
</table>
When (not) to use regular languages

Regular languages are useful for
• processes that require a finite number of steps
• parsing text that doesn’t require us to remember input
  please don’t use regular expressions to parse HTML!

Regular languages are not useful for
• modeling the full power of a computer