Computers: What can’t they do!

Part 3: State Machines Concluded

April 16–17, 2012

CS 60: Principles of Computer Science
Upcoming Schedule

Assignment 10 (Seam Carving) due Monday, April 16

Assignment 11 (Floyd-Warshall, Computability) due Monday, April 23

Mini-Assignment 12 (Uncomputability) due Friday, April 27
## Remaining CS courses: CS 5 and 60 again, more slowly!

<table>
<thead>
<tr>
<th>Math 55 (Discrete Math) ✓</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CS 70 ✓</strong> Data Structures &amp; Program Devel.</td>
</tr>
<tr>
<td><strong>CS 81</strong> Computability and Logic</td>
</tr>
<tr>
<td><strong>CS 105</strong> Computer Systems</td>
</tr>
<tr>
<td><strong>CS 121</strong> Software Development</td>
</tr>
<tr>
<td><strong>CS 131</strong> Programming Languages</td>
</tr>
<tr>
<td><strong>CS 140</strong> Algorithms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>+Three Electives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CS 132</strong> Compilers</td>
</tr>
<tr>
<td><strong>CS 151</strong> Artificial Intelligence</td>
</tr>
<tr>
<td><strong>CS 152 ✓</strong> Neural Networks</td>
</tr>
<tr>
<td><strong>CS 155</strong> Graphics</td>
</tr>
<tr>
<td><strong>CS 181A</strong> Complexity Theory</td>
</tr>
<tr>
<td><strong>CS 181C ✓</strong> Domain-Specific Languages</td>
</tr>
<tr>
<td><strong>CS 181E ✓</strong> Foundations of Parallel Prog.</td>
</tr>
</tbody>
</table>

+Clinic
There are problems that no computer can solve.

What is a problem?
✓ Input: a finite sequence of symbols (a “string”)
✓ Output: yes or no.

What is a computer?
✓ For now, a finite state machine.
Our first Idealized Computer

A Finite State Machine has

✓ A finite set: possible states (configurations)
✓ Rules: how the system transitions between states
  ▶ Depends on current state and current input
✓ Accept or reject, based on the final state

**Deterministic Finite State Machines (DFAs)**

Every state has one transition for every possible input symbol.

Often represented graphically:
**DFA Example**

A DFA that accepts $L = \{ w \in \{0, 1\}^* \mid w$’s **third** character is a 1 $\}$?

Does this DFA accept $010101$? $011010$?
Another DFA Example

A DFA that accepts
$L = \{ w \in \{0, 1\}^* \mid w's \text{ third-to-last character is a 1} \}$?

Does this DFA accept 010101? 011010?
But Are Computers Really Deterministic?

We plan to show that there are problems no DFA can solve.

But maybe determinism is the limiting factor?
DFAs + Nondeterminism = NFAs

An NFA has the magic power to “guess” the right way to go. It can even change states without reading input (“ε transitions”).

Formally: NFA accepts iff there is at least one accepting path.

Does this NFA accept 010101? 011010?
What's Accepted? (bb? b? aaab? ba?)
The following are equivalent:

1. There is a DFA accepting the set \( L \).
2. There’s an NFA accepting \( L \) [Rabin and Scott].
3. \( L \) can be described by a regular expression [Kleene].

In this case, \( L \) is called a Regular Language.
**Regular Expressions do Pattern-Matching**

1* | 1(00)*

\[
\{ w \in \{0, 1\}^* \mid w \text{ is a sequence of (zero or more) ones} \} \\
\cup \\
\{ w \in \{0, 1\}^* \mid w \text{ is a } 1 \text{ followed by an even number of zeros.} \}
\]

------

\[ b(\text{ea|a})(r|d) \]

\[
\{ \text{bear, bead, bar, bad} \}
\]

------

\[ (0|1)(0|1)1(0|1)^* \]

\[
\{ w \in \{0, 1\}^* \mid w\text{'s third character is a } 1 \}
\]
**Regular Expression Ingredients**

Regular expressions are built from:

- $\emptyset$
- $\epsilon$
- A single character from the alphabet (e.g., $a$ or $0$)
- Concatenation: $r_1 r_2$
- Union: $r_1 \mid r_2$
- Repetition (Kleene Star): $r_1^*$
Can this NFA be simulated by a DFA?
Turn this Regular Expression into an NFA

1* | 1(00)*

CS 81 covers two ways to convert NFAs/DFAs into regular expressions
**PRACTICAL APPLICATIONS**

Regular expressions are compact but powerful.

Often used to search/process text

✓ E.g., `egrep`
✓ E.g., Assignment 9 (Unicalc Tokenizer)
✓ E.g., Almost everything written in `perl`

How to implement general regular-expression matching?

✓ Convert RE to a finite state machine
✓ Run input string(s) through!
DFA SIZES

How can we know whether a DFA is as small as possible?

A DFA is minimal if every state is necessary (all states are distinguishable)
The Plan

1. Given languages, find lower bounds on the number of states in any corresponding DFA.
   - E.g., \( \{ a^{3n} \mid n \geq 0 \} \) requires at least 3 states.
   - E.g., \( \{ w \in \{0, 1\}^* \mid w’s \text{third-to-last} \text{ character is a } 1 \} \) requires at least 8 states.

2. Find languages requiring \( \geq \infty \) states
   (are accepted by no finite state machine).

3. If computers are finite state machines,
   no computer can solve those problems!
STATE = FATE!

Suppose we have a state machine for some language. Once two strings $w_1$ and $w_2$ lead to the same state, their future is indistinguishable.
State = Fate: The Contrapositive

If two strings lead to the same state, then the futures beyond this point are indistinguishable.

So, …if two strings in a language have distinguishable futures, they cannot lead to the same state.

Example: Suppose we have a state machine for

\[ L = \{ a^{3n} | n \geq 0 \}. \]

Show that \( a, aa, \) and \( aaa \) must take us to three different states.

Corollary: any state machine for \( L \) has at least 3 states!
Distinguishability Theorem

Given a language \( L \), strings \( w_1 \) and \( w_2 \) are **distinguishable** if

there exists \( z \in \Sigma^* \) such that \( w_1 z \in L \) and \( w_2 z \notin L \) (or vice versa)

A set \( S = \{w_1, w_2, \ldots\} \) is **pairwise distinguishable** if

\( w_i \) and \( w_j \) are distinguishable for all \( i \neq j \).

Theorem

If there is a pairwise distinguishable set of \( N \) strings for a language \( L \), then a DFA accepting \( L \) must have \( \geq N \) states.
More Distinguishability Practice

\[ L = \{0^i1^j \mid i \text{ is even, } j \text{ is odd} \} \]

Are the following distinguishable?

1. 00 and 0
2. 0000 and 00
3. 010 and 1
4. \( \varepsilon \) and 001
**EXAMPLE**

Prove that any DFA recognizing

\[ L = \{ w \in \{0, 1\}^* \mid \text{w's third character is a 1} \} \]

must have at least 5 states.

Proof: \{ ε, 1, 0, 11, 10 \} Proof: \{ ε, 0, 00, 000, 001 \}

**Theorem**

*If there is a pairwise distinguishable set of \( N \) states for a language \( L \), then a DFA accepting \( L \) must have \( \geq N \) states.*
**Nonregular Language Theorem**

A language is said to be **regular** if it can be recognized by some DFA.

Consider the language

\[ L = \{ 0^n 1^n \mid n \geq 0 \} \]

To show it's not regular, …
NONREGULAR LANGUAGE THEOREM

A language is said to be **regular** if it can be recognized by some DFA.

Consider the language of correct multiplications of natural numbers:

\[
L = \{ \quad 0 \times 0 = 0, \quad 0 \times 1 = 0, \quad \ldots \\
\quad 1 \times 0 = 1, \quad 1 \times 1 = 1, \quad \ldots \\
\quad 2 \times 0 = 1, \quad 2 \times 1 = 2, \quad \ldots 
\}
\]

What is the alphabet?
Prove that this language is not regular.
**Conclusion**

*We can identify problems that no finite state machine can solve!*

- ✓ Correct multiplications
- ✓ $L = \{ 0^n1^n \mid n \geq 0 \}$
- ✓ $L = \{ w \mid w \text{ is a palindrome} \}$

If computers are finite state machines, no computer can solve these problems!

The End (?)
Moving On...

Any computer is a finite state machine.

Hence, there are questions it provably cannot answer. (E.g., “is the input a palindrome?”)

But it sure seems like computers can answer these questions (until we run out of memory), so maybe this isn’t so helpful in practice?

Perhaps we should consider more powerful models of computation. (But not too powerful! Why?)