Computers: What can’t they do!
Computers: What can’t they do?

Part 1: Decision Problems and State Machines and Decision Problems

November 17, 2011
CS 42: Principles and Practice of Computer Science

Let what you say be simply ‘Yes’ or ‘No’; anything more than this comes from evil.

Matthew 5:37 (English Standard Version)

It from bit. Otherwise put, every it—every particle, every field of force, even the space-time continuum itself—derives its function, its meaning, its very existence entirely ... [from] answers to yes or no questions, binary choices, bits.

John Archibald Wheeler
**THEOCOMP**

*Computability:* What can and can’t we compute?

*Complexity:* How fast can we compute it?
Computability

✓ How can I prove that a computer can solve some specific problem?
  ▶ E.g., sorting or the Traveling Salesperson Problem

✓ How can I prove that no computer can solve a specific problem?
  ▶ If I prove it today, will it still be true tomorrow?
Describing the Problems to be Solved

Assumption 1: Inputs are finite strings from some finite alphabet of characters.

✓ "37"
✓ "3*7=21"
✓ "sort([3,1,4,1,2], [1,2,3,4])."

Assumption 2: We care about decision problems (i.e., answer is just yes or no)

✓ Is 37 prime?
✓ Does $3 \times 7 = 21$?
✓ Is [1, 2, 3, 4] the result of sorting [3, 1, 4, 1, 2]?
✓ :  

Justifying the Assumptions

Assumption 1: Inputs are finite strings from a finite alphabet of characters.
   ✓ Generalizes “everything’s just a string of bits”

Assumption 2: We care about decision problems (i.e., answer is just yes or no)
   ✓ We can solve other problems by asking enough questions.
   ✓ E.g., how could I figure out $3 \times 7$ by asking yes/no questions?
Combining the Assumptions

Any decision problem is equivalent to asking

“Is the input a member of the set \( L \)?”

for a suitable set \( L \).

✓ For primality testing, \( L = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots \} \).

✓ For multiplication-correctness, \( L = \)?
Conclusion

A one-to-one correspondance:

(computational) problems to solve

↔

sets of finite strings.

These sets are called “formal languages”
An alphabet $\Sigma$ is a finite, nonempty set. The elements of $\Sigma$ are called letters or symbols.
Formal Languages: Strings

A string over $\Sigma$ is a sequence

$c_1c_2 \cdots c_n$

where $n \geq 0$ and each $c_i \in \Sigma$.

✓ Every string is finite, but could be arbitrarily long!
✓ We will write strings without quotation marks:
  
  $xyzzy$

✓ We write the empty string as $\varepsilon$ or $\lambda$

Note: $\varepsilon, \lambda \notin \Sigma$!
LANGUAGES

A “language $L$ over $\Sigma$” is a set of strings over $\Sigma$.

✓ The set of all strings over $\Sigma$ is written $\Sigma^*$.
✓ The empty set $\emptyset$ is a language.
✓ The non-empty set $\{\varepsilon\}$ is a language.
✓ Languages may be infinite, even if they contain only finite strings!
✓ Other examples?
A First Uncomputability Result

✓ Every computer program can be represented as a string in \( \{0, 1\}^* \).
✓ There are as many string/programs as there are natural numbers
✓ There are as many languages as there are real numbers!

If you choose a language “at random,” what is the probability it can be decided by some program?
Idealized Computers

We need a precise (mathematical) definition, abstracting away details that might change.

✓ Operating System
✓ Processor speed
✓ Memory capacity
✓ Power source (electricity, natural gas, dilithium, ...)
✓ Construction materials (silicon, graphene, legos, ...)
✓ Programming language (Java, Racket, Prolog, HMMM, ...)
✓ Architecture (single core, multicore, manycore, GPU, VLIW, ...)
✓ Data representation (ASCII, Unicode, binary, trinary, ...)

Ideal Computer Systems, Inc.
Today’s Idealized Computer

State Machine, which

✓ A set of possible “configurations” (states)
✓ Rules for how the system proceeds from one state to another
  ▶ Depends on current state and current input
✓ Accept or reject, based on the input this far.
What is a state?

The state of a system at some instant consists of the all internal information needed to figure out how to proceed.

Many systems have more than 50 states!
A finite state machine is a state machine with finitely many possible states.

http://www.flickr.com/photos/alexbrn/5035170693

http://www.flickr.com/photos/cambodia4kidsorg/3019948738
Deterministic Finite State Machines (DFAs)

Every state has one transition for every possible input symbol. Often represented graphically:

Does this machine accept 100? 1010?
What is the language of this machine?
Finite states = finite memory. What are we “remembering” in each state?
DFAs are everywhere!
There's a lot missing here!

How many states total in this kind of lock?
DFAs are everywhere!

FSM == FearSoMe?

The FSM controlling Shambler monsters…

I'm Quaking in my AstroBoots.
DFAs are everywhere!

Computers: State changes over time
**Another DFA**

What is the language of this machine?

![DFA diagram](image)

Finite states = finite memory. What are we “remembering” in each state?
“Quiz”

NAME:

What DFA accepts \( L = \{ w \in \{0, 1\}^* \mid w \text{ is a binary number that’s even} \} \)

DFA for \( L = \{ w \in \{0, 1\}^* \mid w \text{ does not contain the substring 101} \} \)?
More Examples

What DFA accepts

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

What DFA accepts

\[ L = \{ w \in \{0, 1\}^* \mid w's \text{third-to-last} \text{ character is a } 1 \} \]
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)
**Distinguishing Strings**

Recall

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

Meet our four string couples for today:

<table>
<thead>
<tr>
<th>Couple #1</th>
<th>Couple #2</th>
<th>Couple #3</th>
<th>Couple #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{w}_1 = 011 )</td>
<td>( \mathbf{w}_1 = 011 )</td>
<td>( \mathbf{w}_1 = 01 )</td>
<td>( \mathbf{w}_1 = 0 )</td>
</tr>
<tr>
<td>( \mathbf{w}_2 = 111 )</td>
<td>( \mathbf{w}_2 = 110 )</td>
<td>( \mathbf{w}_2 = 10 )</td>
<td>( \mathbf{w}_2 = 10 )</td>
</tr>
</tbody>
</table>

If we continue on with more digits, are the “fates” of these couples the same or different?
**String Matchmaking, Formalized**

Given a language $L \subseteq \Sigma^*$

Two 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 $S = \{w_1, \ldots, \ldots\}$ is **pairwise distinguishable** with respect to language $L$ if

$w_i$ and $w_j$ are distinguishable for all $i \neq j$. 
More Distinguishability Practice

\[ L = \{ 0^i 1^j \mid i \text{ is even, } j \text{ is odd} \} \]

Are the following distinguishable?

1. 00 and 0
2. 0000 and 00
3. 010 and 1
4. \lambda and 001
**Distinguishability Theorem**

Key insight: if $w_1$ and $w_2$ are distinguishable, they cannot both lead to the same state in our state machine.

**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.*
EXAMPLE

Prove that any DFA recognizing

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

must have at least 5 states.

Proof: \{ \lambda, 1, 0, 11, 10 \} Proof: \{ \lambda, 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.
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:

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

(What is the alphabet?)

Prove that this language is not regular.
We can identify problems that no finite state machine can solve.

✓ $L = \{ 0^n1^n \mid n \geq 0 \}$

✓ $L = \{ ww \mid w \in \{0, 1\}^* \}$

If computers are finite state machines, then no computer can solve them either!
A HMC DFA (Author Unknown)

Complete the finite-state machine below with at least three states and transitions of your own design. Feel free to ignore or modify the sample states/ transitions provided.