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

November 23, 2011

CS 81: Computability & Logic

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 sort $[3, 1, 4, 1, 2]$ 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, $\mathrm{L} = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$.
- ✓ For multiplication-correctness,

$$\mathrm{L} = \{ \quad 0\times0=0, \;\; 0\times1=0, \;\; \ldots \\ 1\times0=1, \;\; 1\times1=1, \;\; \ldots \\ 2\times0=1, \;\; 2\times1=2, \;\; \ldots \\ \}$$

Conclusion

A one-to-one correspondance:

(computational) problems to solve

sets of finite strings.

These sets are called “formal languages”

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, . . .)

Idealized Computers

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.

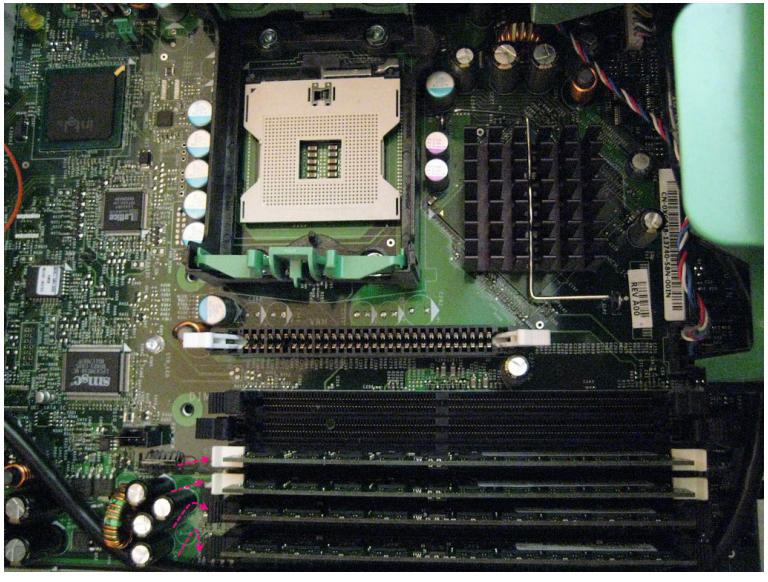
What is a state?

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

State?

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

State?



http://www.flickr.com/photos/cambodia4kidsorg/3019948738

Many systems have more than 50 states!

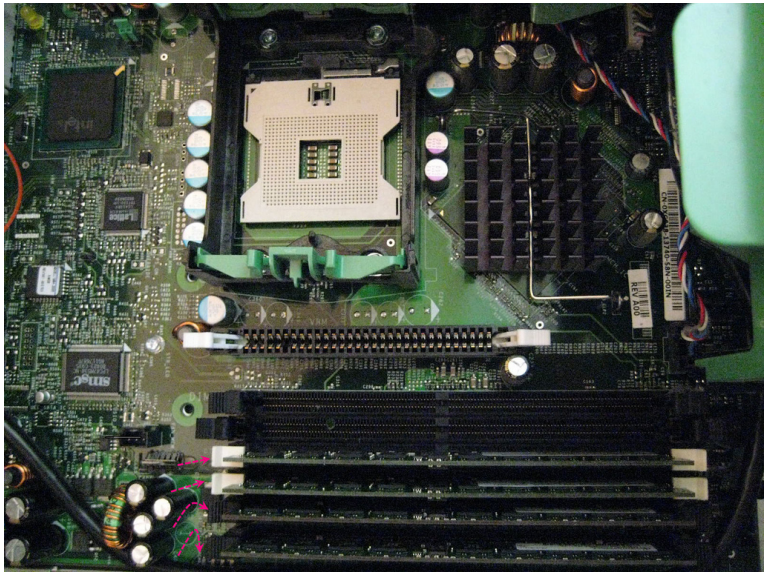
Finite State Machines

A *finite state machine* is a state machine with finitely many possible states.

Finite State?

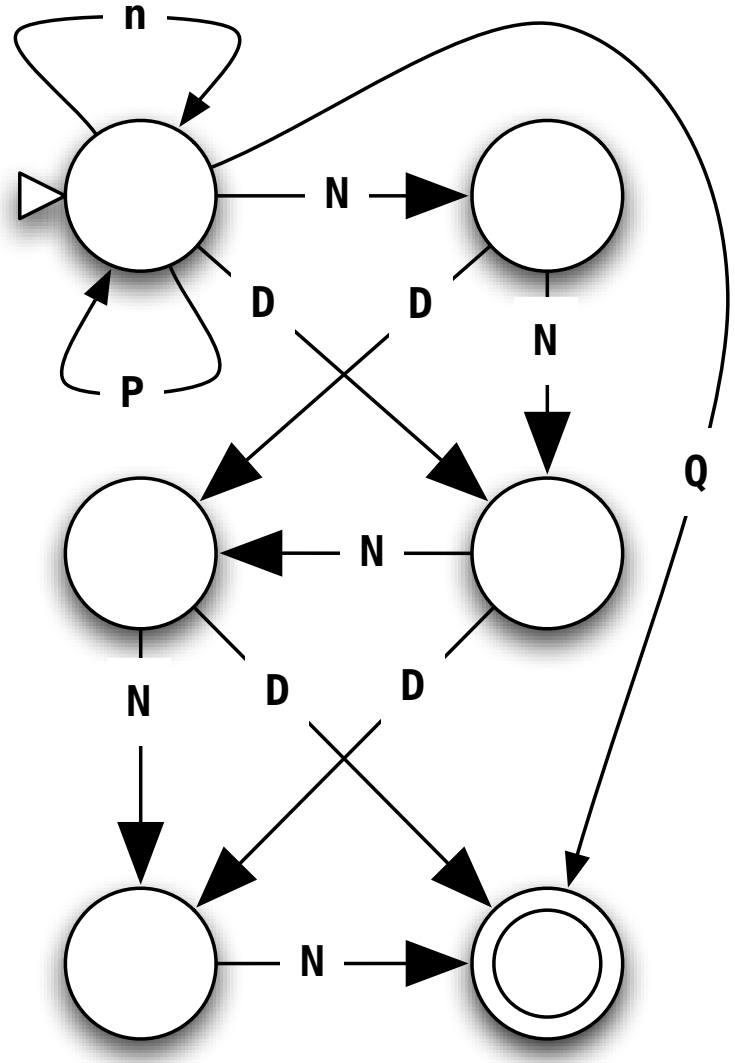
http://www.flickr.com/photos/alexbrn/5035170693

Finite State?



http://www.flickr.com/photos/cambodia4kidsorg/3019948738

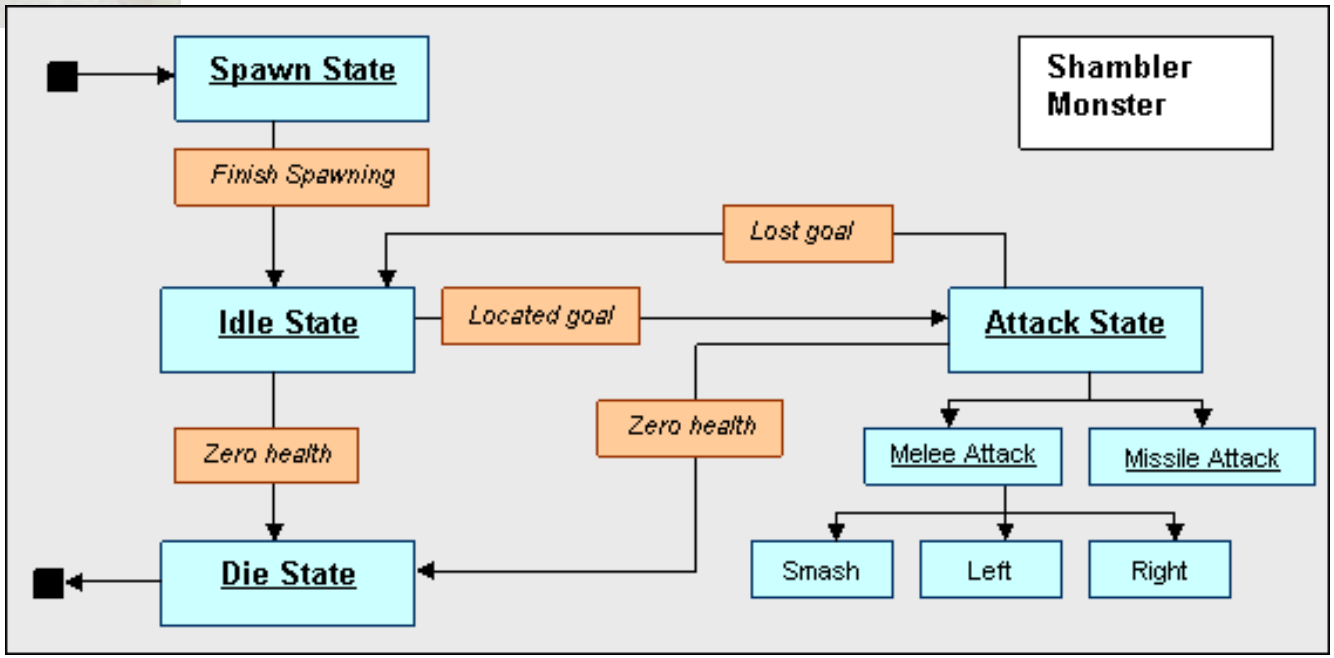
DFAs are everywhere!



DFAs are everywhere!

FSM == FearSoMe?

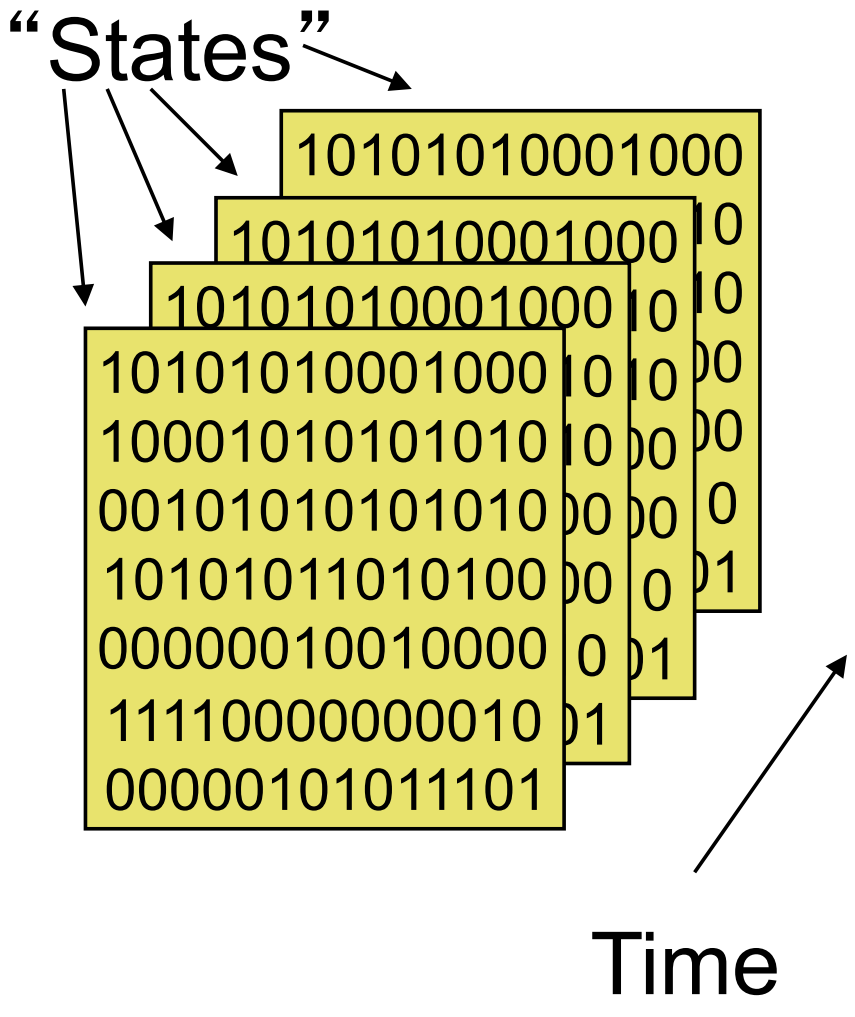
The FSM controlling
Shambler monsters…



I'm ***Quaking*** in my AstroBoots

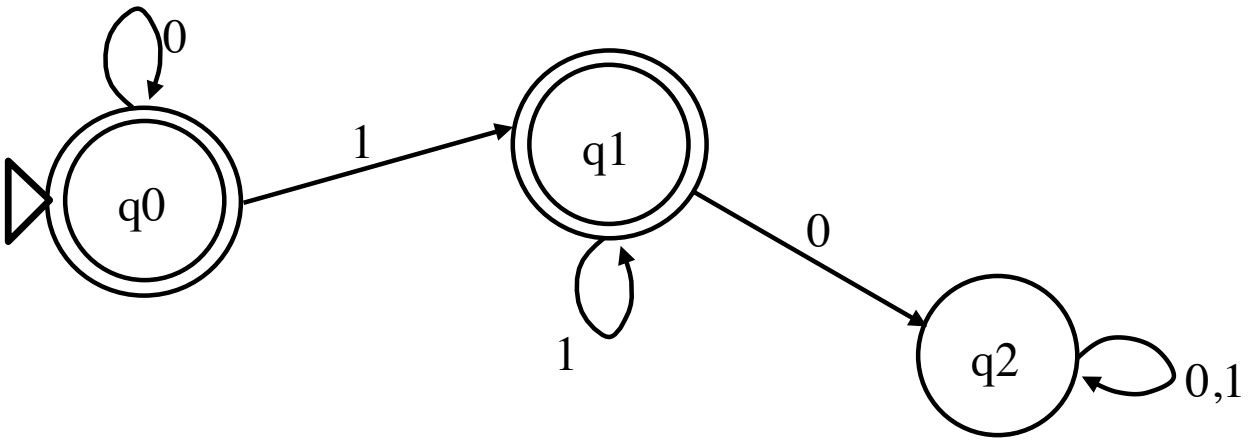
DFAs are everywhere!

Computers: State changes over time



Another DFA

What is the language of this machine?



Finite states = finite memory. What are we "remembering" in each state?

A Jewel of Theoretical Computer Science



The following are equivalent:

- ✓ There is a DFA accepting the language L.
- ✓ [Rabin and Scott] There's an NFA accepting L.
- ✓ [Kleene] L can be described by a regular expression.

REs to the Rescue!

← PERL — practical extraction and report language

Email Address

```
\b[A-Z0-9._%-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b
```

Options: case insensitive

www.regular-expressions.info/regexbuddy/email.html

Google /* Code Search */ LABS

Google Code Search

About Google Code Search

Frequently Asked Questions

1. What kind of code are you crawling?
2. What regexp syntax does Code Search support?
3. What programming languages do you support?

but how does regular expression matching actually ***work***? →

Regular Expression Ingredients

A regular expression s can be:

- ✓ $\emptyset$
- ✓ ϵ (or λ)
- ✓ A single character from the alphabet (e.g., a or 0)
- ✓ Concatenation: $r_1 r_2$
- ✓ Union: $r_1 \cup r_2$
- ✓ Repetition (Kleene Star): r_1^*.

Regular Expressions in Practice

Unix's **egrep** *command does* line-by-line *search for* text *matching a* regular *expression.*

```
egrep   'hh'        /usr/share/dict/words
egrep   'y.*y'      /usr/share/dict/words
egrep   '(xq|hq)'   /usr/share/dict/words
egrep   '^y.*y$'    /usr/share/dict/words
```

Many *systems, including* **egrep** *add extra operations* (mostly, but not entirely, *abbreviations*)

Distinguishing Strings

Recall

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

.

Meet our four string couples for today:

Couple #1	Couple #2	Couple #3	Couple #4
$w_1 = 011$	$w_1 = 011$	$w_1 = 01$	$w_1 = 0$
$w_2 = 111$	$w_2 = 110$	$w_2 = 10$	$w_2 = 10$

If we continue on with more digits (same suffix for both strings in a couple!), are the "fates" of these couples the *same* or *different*?

String Matchmaking, Formalized

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

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. ε and 001

Distinguishability Theorem

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

Theorem

If there is a pairwise distinguishable set of N *states for a language* L*, then a DFA accepting* L *must have* $\geqslant N$ *states.*

Example

Prove that any DFA recognizing

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

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 $\geqslant N$ states.

Nonregular Language Theorem

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

Consider the language

$$L = \{ 0^n 1^n \mid n \geqslant 0 \}$$

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:

$$L = \{ \quad 0\times0=0,\ \ 0\times1=0,\ \ \ldots \\ 1\times0=1,\ \ 1\times1=1,\ \ \ldots \\ 2\times0=1,\ \ 2\times1=2,\ \ \ldots \\ \}$$

(What is the alphabet?)

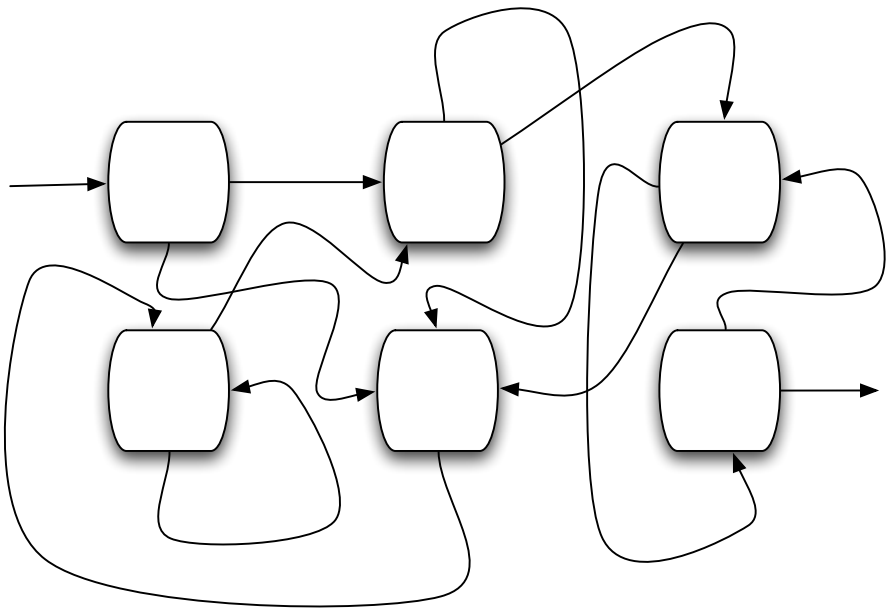
Prove that this language is not regular.

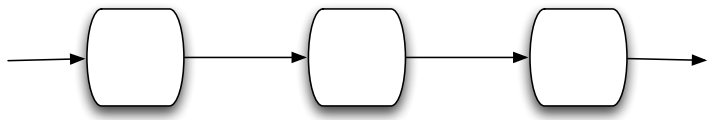
Finite Maze Theorem

For every finite maze there is a number $\mathbf{p}$, such that

For every path through the maze $\mathbf{s}$ with $|\mathbf{s}| \geqslant \mathbf{p}$:

- The path $\mathbf{s}$ contains at least one loop, which starts and ends within the first $\mathbf{p}$ steps.
- There are infinitely many paths through the maze(at least one shorter, and arbitrarily many longer) whose lengths differ by a multiple of some constant.





A Pumping Lemma (There are others)

If L is a regular language, then

there exists a number p such that

For every $s \in L$ with $|s| \geqslant p$

we can decompose s into xyz where

1. $y \neq \varepsilon$
2. $|xy| \leqslant p$
3. $xy^iz \in L$ for every $i \geqslant 0$.

To show that

$$L = \{ 0^n 1^n \mid n \geqslant 0 \}$$

is not regular, ...

Conclusions

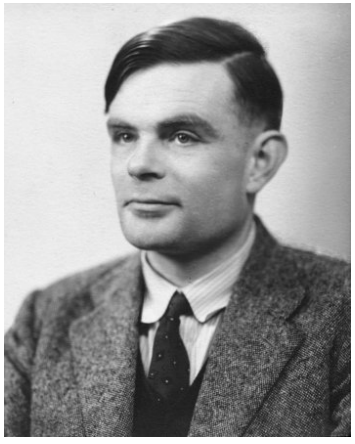
Any possible computer is a finite state machine.

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?)

Alan Turing



WWII

Enigma machine ~ The axis's encryption engine

1946

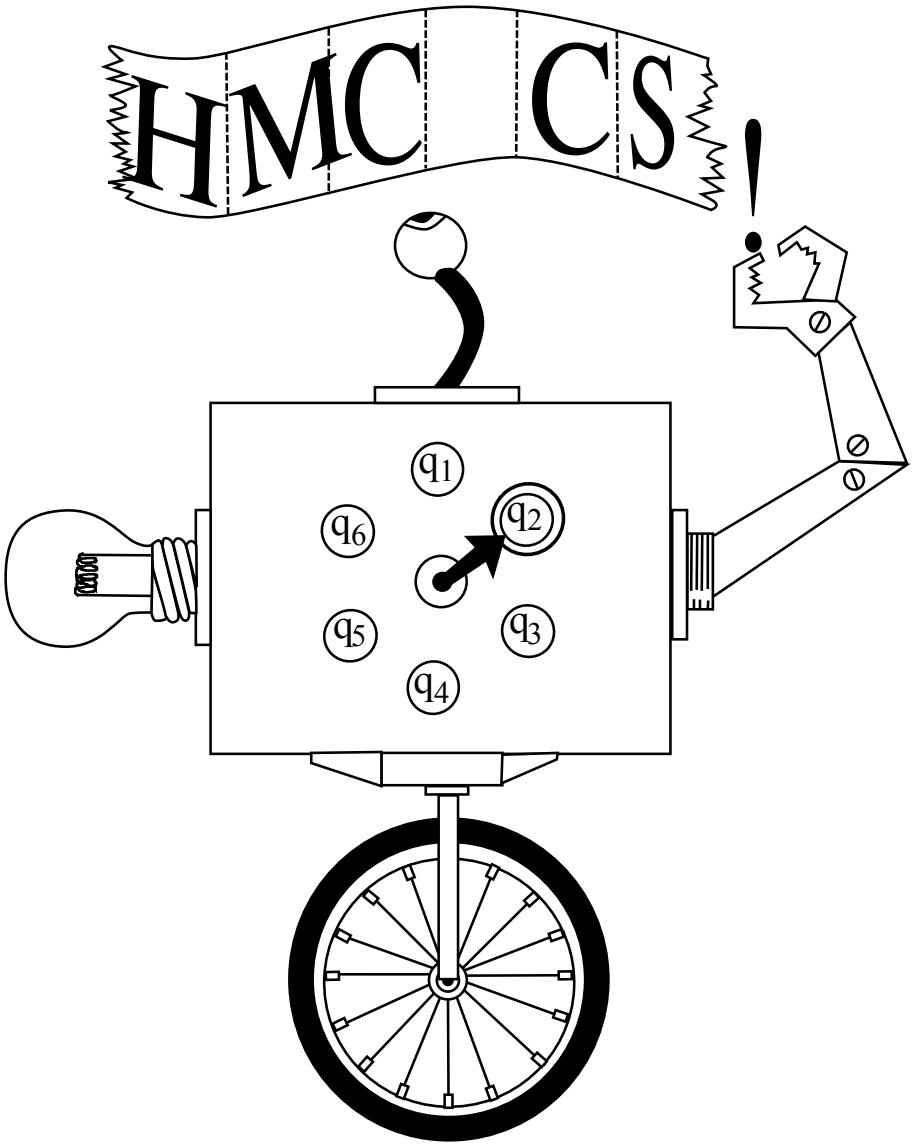
2007
Bletchley Park

Turing Machines

A simple model of universal computation.

- ✓ Control/Program: a finite state machine
- ✓ Data: An *infinite* tape, with a read/write head. Initially blank except for the input string.
- ✓ Steps determined by current state + current symbol;
  Can overwrite current symbol, and move left or right.
- ✓ Can stop (and say yes/no) or not.

Turing Machine: Artist's Conception



Why Turing Machines?

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes

Why Turing Machines?

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent†. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as

Why Turing Machines?

17 or 999999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of symbols). The differences from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be "arbitrarily close" and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

Let us imagine the operations performed by the computer to be split up

TM Variations

The following yield no extra power:

- ✓ Adding the option to write or not on each step.
- ✓ Adding the option to stay-in-place rather than moving L/R.
- ✓ Adding an extra "Erase Tape" move.
- ✓ Finitely many tapes with finitely many (independent) read/write heads
- ✓ 2-D infinite tape
- ✓ Nondeterminism (given a choice, "magically" pick the right alternative)

So…

Can TMs solve all problems? No!

- ✓ By cardinality, practically no language corresponds to a TM.
- ✓ Halting is not decidable, but is semidecidable
- ✓ Not-Halting is neither decidable nor semidecidable
- ✓ Rice's Theorem: *nontrivial* questions about a given TM's *language* are not decidable. (May or may not be semidecidable)
- ✓ Reductions can show other questions are

So…

Is there something more powerful than a Turing Machine? Well, yes. But…

Church-Turing Thesis: any plausible model of computation is no more powerful than a Turing Machine.

- ✓ Proof: by lack of counterexample.