A Deterministic TM consists of

✓ A finite set \( Q \) of control states
✓ A finite alphabet \( \Sigma \)
✓ A finite “tape alphabet” \( \Gamma \) \((\Sigma \subset \Gamma, \sqcup \in \Gamma \setminus \Sigma)\)
✓ A starting state \( q_0 \in Q \)
✓ Accepting/Rejecting (halting) states \( q_{\text{accept}}, q_{\text{reject}} \in Q \)
✓ Transitions \( \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \)
TM Variations

The following yield no extra power:

✓ Adding the option to write or not
✓ Adding the option to stay-in-place rather than moving L/R.
✓ Making the tape infinite in both directions
✓ Adding an extra "Erase Tape" move.
✓ Multiple tapes with multiple (independent) read/write heads
✓ Finitely many registers with finite capacity
✓ 2-D infinite tape
✓ Nondeterminism (!)

Many attempts to define models of computation; all turn out to be equivalent to Turing Machines.

✓ If you can do it in C++, a TM can do it (slowly, encodedly)
TMs as Strings

A TM can be described by the finite list of transition rules:

\[
\begin{align*}
q_0, \_ & \rightarrow q_1, \_, R \\
q_1, \_ & \rightarrow q_0, \_, L \\
q_1, x & \rightarrow q_2, x, L \\
q_2, a & \rightarrow q_1, x, R
\end{align*}
\]

We can devise a way to encode an arbitrary set of such rules into a fixed alphabet. Although the number of states and tape symbols can be arbitrary

- we can encode these by using strings of symbols like \( \{0, 1\} \)
- concatenate the symbols to describe rules
- concatenate rules to describe machines.
**Configurations as Strings**

A configuration can be expressed as a finite string, *e.g.*, 

011q01 □110

means

✓ The contents of the tape is 01101 □110 (padded on the right with blanks)
✓ The TM is in control state *q*
✓ The TM is pointing to the second 0.
Universal Turing Machines

UTMs can be shown to exist by constructing them. Think about what would be required.

✓ The tape has to hold the tape of the machine being simulated.
✓ The tape has to hold the program of the machine being simulated.
✓ The tape has to hold the current state of the machine being simulated.

All this is possible, if somewhat laborious to construct.
Specific UTMs

✓ The first was constructed by Turing himself.
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
✓ Minsky (1960) gave a 7-state 6-symbol machine.
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
✓ Minsky (1960) gave a 7-state 6-symbol machine.
✓ Watanabe (1961) gave an 8-state 5-symbol machine.
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
✓ Minsky (1960) gave a 7-state 6-symbol machine.
✓ Watanabe (1961) gave an 8-state 5-symbol machine.
✓ Minsky (1962) gave a 7-state 4-symbol machine.
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
✓ Minsky (1960) gave a 7-state 6-symbol machine.
✓ Watanabe (1961) gave an 8-state 5-symbol machine.
✓ Minsky (1962) gave a 7-state 4-symbol machine.
✓ Rogozhin (1996) gave a 4-state 6-symbol machine.
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
✓ Minsky (1960) gave a 7-state 6-symbol machine.
✓ Watanabe (1961) gave an 8-state 5-symbol machine.
✓ Minsky (1962) gave a 7-state 4-symbol machine.
✓ Rogozhin (1996) gave a 4-state 6-symbol machine.
✓ Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
**Specific UTMs**

- The first was constructed by Turing himself.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
- Minsky (1960) gave a 7-state 6-symbol machine.
- Watanabe (1961) gave an 8-state 5-symbol machine.
- Minsky (1962) gave a 7-state 4-symbol machine.
- Rogozhin (1996) gave a 4-state 6-symbol machine.
- Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
- Smith and Wolfram (2007) gave a 2-state 3-symbol machine.
- No 7-state 7-symbol UTM exists.
Specific UTMs

✓ The first was constructed by Turing himself.
✓ Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine (with lots more states or symbols, respectively).
✓ Minsky (1960) gave a 7-state 6-symbol machine.
✓ Watanabe (1961) gave an 8-state 5-symbol machine.
✓ Minsky (1962) gave a 7-state 4-symbol machine.
✓ Rogozhin (1996) gave a 4-state 6-symbol machine.
✓ Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
✓ Smith and Wolfram (2007) gave a 2-state 3-symbol machine.
✓ No 2-state 2-symbol UTM exists.
A Specific UTM

2-state, 5 symbol UTM published by Wolfram in 2002

Turing Machines as Enumerators

Several variant definitions. Each specify a language \( L \).

1. A TM that prints out all the members of \( L \), one at a time (but not necessarily in any particular order)
2. A TM that prints out all the members of \( L \), one at a time (but...) with arbitrarily many repeats.
3. A TM that, given an integer \( n \), returns the \( n \)th element of a sequence like (1) above.
4. A TM that, given an integer \( n \), returns the \( n \)th element of a sequence like (2) above.

For a fixed language, all these are interconvertable.

Theorem

A language is recognizable \( \iff \) it can be enumerated.
Computability and Uncomputability

April 13, 2011
CS 81: Computability and Logic
QUESTION

How is my iPad more like a Finite State Machine than like a Turing Machine?

How is my iPad more like a Turing Machine than like a Finite State Machine?
Church-Turing Thesis

If it can be done at all, then it can be done by

✓ A Turing Machine
✓ Lambda Calculus
✓ An Unrestricted Grammar
✓ A 2-register machine
✓ C
✓ ...

(Note: assumes suitably coded inputs and outputs)
Is There More?

Decidable
Context Free
Regular
\(a^*b^*\)
\(a^n b^n\)
\(a^p b^p c^p\) (p perfect)

?
Some Languages Aren’t Decidable

Given a finite $\Sigma$, how many strings are there?
Some Languages Aren’t Decidable

Given a finite $\Sigma$, how many strings are there?

Given a finite $\Sigma$, how many languages are there?
Some Languages Aren’t Decidable

Given a finite $\Sigma$, how many strings are there?

Given a finite $\Sigma$, how many languages are there?

How many TMs are there?
Some Languages Aren’t Decidable

Given a finite $\Sigma$, how many strings are there?

Given a finite $\Sigma$, how many languages are there?

How many TMs are there?

QED
But Wait…

How many languages over $\Sigma$ could I describe (say, in a \textsc{LaTeX} document)?
But Wait…

How many languages over $\Sigma$ could I describe (say, in a \LaTeX\ document)?

How many TMs are there?
But Wait…

How many languages over \( \Sigma \) could I describe (say, in a LaTeX document)?

How many TMs are there?

QED?
LANGUAGES OF ACCEPTANCE

Which are recognizable (by a TM)? Decidable?

✓ $A_{DFA} = \{ \langle D, w \rangle \mid D \text{ a DFA, } D \text{ accepts } w \}$

✓ $A_{NFA} = \{ \langle N, w \rangle \mid N \text{ an NFA, } N \text{ accepts } w \}$

✓ $A_{RE} = \{ \langle R, w \rangle \mid R \text{ a regexp, } R \text{ matches } w \}$

✓ $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ a CFG, } G \text{ produces } w \}$

✓ $A_{TM} = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ accepts } w \}$
Digression: Bootstrap a Compiler

Lots of compilers are written in the same language they compile!
Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

✓ Gnu C Compiler (used in CS 105) is written in C
Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

✓ Gnu C Compiler (used in CS 105) is written in C
✓ Glasgow Haskell Compiler (used in CS 131) is in Haskell etc.
Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!
✓ Gnu C Compiler (used in CS 105) is written in C
✓ Glasgow Haskell Compiler (used in CS 131) is in Haskell etc.

Practical reasons to run programs on their own source code!
\( \mathcal{A}_{TM} \) IS NOT DECIDABLE

<table>
<thead>
<tr>
<th>( \langle M_0 \rangle )</th>
<th>( \langle M_1 \rangle )</th>
<th>( \langle M_2 \rangle )</th>
<th>( \langle M_3 \rangle )</th>
<th>( \langle M_4 \rangle )</th>
<th>( \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_0 )</td>
<td>Acc.</td>
<td>Acc.</td>
<td>Acc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_2 )</td>
<td>Acc.</td>
<td></td>
<td>Acc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_4 )</td>
<td></td>
<td>Acc.</td>
<td>Acc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Is There More?

Recognizable
Decidable
Context Free
Regular
\(a^*b^*\)
\(a^n b^n\)
Decidable
\(a^p b^p c^p\) (p perfect)
Recognizable
\(ATM\)

?
What is the complement of $A_{TM}$?
What is the complement of $A_{TM}$?
Theorem

The language

\[ H = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ halts on } w \} \]

is not decidable.
Obligatory Corollary

Theorem

The language

\[ H = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ halts on } w \} \]

is not decidable.

Proof.
Suppose there were a halt-checking TM…  □