Lecture 9
Turing Machines: Recognizability, Decidability, The Church-Turing Thesis
Turing Machine (1936)

In each step:
- Reads a symbol
- Writes a symbol
- Changes state
- Moves Left or Right

FINITE STATE CONTROL

INFINITE REWRITABLE TAPE

q₁

tape head

“blanks”

INPUT

OUTPUT

...
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...
GREAT. A WAREHOUSE FILLED WITH MILES AND MILES OF REWRITABLE TAPE! WHAT ARE WE EVER GOING TO DO WITH THIS, ALAN?

...ALAN?

And thus the Turing Machine was born.


#inspired
Turing Machines versus DFAs

The input is written on an infinite tape with “blank” symbols after the input.

The “tape head” can move right and left.

The TM can both write to and read from the tape, and can write symbols that aren’t part of input.

Accept and Reject take immediate effect.
A TM for $L = \{ \text{w#w | w } \in \{0,1\}^* \}$ over $\Sigma = \{0,1,\#\}$

STATE
$q_0, f \; q_1, \text{FIND } \#$ $q_#, f \; q_0, f \; q_1, \text{FIND } \square \; q \text{ GO LEFT}$

and so on...

1. If there’s no # on the tape (or more than one #), reject.
2. While there is a bit to the left of #,
   - Replace the first bit $b$ with $X$, and check if the first bit $b’$ to the right of the # is identical to $b$. (If not, reject.)
   - Replace that bit $b’$ with an $X$ too.
3. If there’s a bit to the right of #, then reject else accept
Definition: A Turing Machine is a 7-tuple $\mathcal{T} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \not\in \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
This Turing machine *decides* the language \{0\}.
This Turing machine \textit{recognizes} the language \{0\}

Three kinds of behaviors: accepting, rejecting, and running forever!
Turing Machine Configurations

corresponds to the configuration:

$q_0\overbrace{1101000110}^{\in (Q \cup \Gamma)^*}$
Turing Machine Configurations

$q_1$ corresponds to the configuration:

$0q_110100011001110 \in (Q \cup \Gamma)^*$
Turing Machine Configurations

0000011110 \( q_7 \)

corresponds to the configuration:

0000011110 \( q_7 \) \( \square \) \( \in (Q \cup \Gamma)^* \)
Defining Acceptance and Rejection for TMs

Let $C_1$ and $C_2$ be configurations of a TM $M$

**Definition.** $C_1$ *yields* $C_2$ if $M$ is in configuration $C_2$ after running $M$ in configuration $C_1$ for one step.

**Example.** Suppose $\delta(q_1, b) = (q_2, c, L)$
Then $aq_1bb$ yields $q_2acb$
Suppose $\delta(q_1, a) = (q_2, c, R)$
Then $abq_1a$ yields $abcq_2$ □

Let $w \in \Sigma^*$ and $M$ be a Turing machine.

$M$ *accepts* $w$ if there are configs $C_0, C_1, \ldots, C_k$, s.t.
• $C_0 = q_0w$ [the initial configuration]
• $C_i$ yields $C_{i+1}$ for $i = 0, \ldots, k-1$, and
• $C_k$ contains the accept state $q_{accept}$
A TM $M$ **recognizes** a language $L$ if $M$ **accepts** exactly those strings in $L$

A language $L$ is **recognizable** *(a.k.a. recursively enumerable)* if some TM recognizes $L$

A TM $M$ **decides** a language $L$ if $M$ **accepts** all strings in $L$ and **rejects** all strings not in $L$

A language $L$ is **decidable** *(a.k.a. recursive)* if some TM decides $L$

$L(M) :=$ set of strings $M$ accepts
A Turing machine for deciding \( \{ 0^{2^n} \mid n \geq 0 \} \)

Turing Machine PSEUDOCODE:

1. Sweep from left to right, \( \times \)-out every other 0
2. If in step 1, the tape had only one 0, accept
3. If in step 1, the tape had an odd number of 0’s (at least 3), reject
4. Move the head left to the first input symbol.
5. Go to step 1.

Why does this work?

Observation: Every time we return to step 1, the number of 0’s on the tape has been halved.
\( \{ 0^{2^n} \mid n \geq 0 \} \)

- **Step 1**: 
  - \( 0 \rightarrow \square, R \)
  - \( x \rightarrow x, R \)

- **Step 2**: 
  - \( \square \rightarrow \square, R \)
  - \( \square \rightarrow \square, L \)
  - \( x \rightarrow x, R \)

- **Step 3**: 
  - \( x \rightarrow x, R \)
  - \( \square \rightarrow \square, R \)

- **Step 4**: 
  - \( x \rightarrow x, L \)
  - \( 0 \rightarrow 0, L \)
  - \( 0 \rightarrow x, R \)

States: 
- \( q_0 \): Start state
- \( q_1 \): State after reading the first 0
- \( q_2 \): State for even numbers
- \( q_3 \): State for odd numbers
- \( q_4 \): Accept state
- \( q_{\text{reject}} \): Reject state
\[ \{ 0^{2^n} \mid n \geq 0 \} \]
MULT = \{a^ib^jc^k \mid k = i*j, \text{ and } i, j, k \geq 1\}

TURING MACHINE PSEUDOCODE:

1. If the input doesn’t match $a^*b^*c^*$, \textit{reject}.
2. Move the head back to the leftmost symbol.
3. Cross off one $a$, scan to the right until see $b$.
   Sweep between $b$’s and $c$’s, crossing off one of each until all $b$’s are crossed off.
   If all $c$’s get crossed off while doing this, \textit{reject}.
4. \textbf{Uncross} all the $b$’s.
   If there is some $a$ left, then repeat stage 3.
   If all $a$’s are crossed off,
     Check if all $c$’s are crossed off.
     If yes, then \textit{accept}, else \textit{reject}. 
MULT = \{a^i b^j c^k \mid k = i \times j, \text{ and } i, j, k \geq 1\}

Check matches $a^*b^*c^*$
Cross off an $a$
Cross off one $c$ for each $b$
“Uncross” the $b$’s
Repeat the crossing, until all $a$’s crossed (or reject early)
Accept
Turing Machines are Robust!

Many variants and models can be defined. As long as your favorite model reads and writes a finite number of symbols in each step, it doesn’t matter!

A good ole TM can still simulate it!
Multitape Turing Machines

Finite State Control

\[ \delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L,R\}^k \]
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Nondeterministic Turing Machines

Have multiple transitions for a state, symbol pair

**Theorem:** Every nondeterministic Turing machine $N$ can be transformed into a Turing Machine $M$ that accepts precisely the same strings as $N$. ($L(M)=L(N)$)

**Proof Idea (more details in Sipser p.178-179)**
Pick a natural ordering on the strings in $(Q \cup \Gamma \cup \#)^*$

$M(w)$: For all strings $D \in (Q \cup \Gamma \cup \#)^*$ in the ordering,
Check if $D = C_0 \# \cdots \# C_k$ where $C_0, \ldots, C_k$ is an accepting computation history for $N$ on $w$.
If so, *accept*.
What else can Turing Machines do?

They can analyze and simulate other TMs.

To do that, we need to encode TMs as strings.
Fact: We can encode Turing Machines as bit strings

\[ 0^n 10^m 10^k 10^s 10^t 10^r 10^u 1 \ldots \]

- \( n \) states
- \( m \) tape symbols (first \( k \) are input symbols)
- \( m \) tape symbols
- \( s \) start state
- \( t \) reject state
- \( r \) accept state
- \( u \) blank symbol

\[
( (p, i), (q, j, L) ) = 0^p 10^i 10^q 10^j 101
\]

\[
( (p, i), (q, j, R) ) = 0^p 10^i 10^q 10^j 1001
\]

Can map every TM \( M \) to a bit string \( \langle M \rangle \)
We can also encode DFAs and NFAs as *bit strings*, and $w \in \Sigma^*$ as *bit strings*

For $x \in \Sigma^*$ define $b_\Sigma(x)$ to be its binary encoding.

For $x, y \in \Sigma^*$, define the *pair of x and y* as a binary string encoding both $x$ and $y$

$$\langle x, y \rangle := 0^{\lfloor b_\Sigma(x) \rfloor} 1 b_\Sigma(x) b_\Sigma(y)$$

Then we define the following languages over $\{0,1\}$:

$A_{\text{DFA}} = \{ \langle D, w \rangle \mid D \text{ encodes a DFA over some } \Sigma, \text{ and } D \text{ accepts } w \in \Sigma^* \}$

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

$A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ encodes a TM, } M \text{ accepts } w \}$
Universal Turing Machines

Theorem: There is a Turing machine $U$ which takes as input:
- the code of an arbitrary TM $M$
- and an input string $w$
such that $U$ accepts $\langle M, w \rangle \iff M$ accepts $w$.

This is a *fundamental* property of TMs:
There is a Turing Machine that can run arbitrary Turing Machine code!

Note that DFAs/NFAs do *not* have this property.
That is, $A_{\text{DFA}}$ and $A_{\text{NFA}}$ are not regular.
Want: $U$ accepts $\langle M, w \rangle \iff M$ accepts $w$.

Can make a multitape TM $U$ with four tapes:
1. Input tape: receives $\langle M, w \rangle$
2. State tape: holds the current state of $M$
3. Machine code tape: holds transitions of $M$
4. Simulation tape: content is identical to $M$’s tape

For each step of $M$: $U$ looks up the matching transition in machine code tape, updates the state and simulation tape
\[ A_{\text{DFA}} = \{ \langle D, w \rangle \mid D \text{ is a DFA that accepts string } w \} \]

**Theorem:** \( A_{\text{DFA}} \) is decidable

**Proof:** A DFA is a special case of a TM. Run the universal \( U \) on \( \langle D, w \rangle \) and output its answer!

\[ A_{\text{NFA}} = \{ \langle N, w \rangle \mid N \text{ is an NFA that accepts string } w \} \]

**Theorem:** \( A_{\text{NFA}} \) is decidable. (Why?)

\[ A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \} \]

**Theorem:** \( A_{\text{TM}} \) is recognizable (Why?)
The Church-Turing Thesis

Everyone’s Intuitive Notion of Algorithms = Turing Machines

This is not a theorem – it is a falsifiable scientific hypothesis.

And it has been thoroughly tested!
Thm: There are *unrecognizable* languages

Assuming the Church-Turing Thesis, this means there are problems that *no* computing device will *ever* solve!

We will prove there is no *onto* function from the set of all Turing Machines to the set of all languages over \{0,1\}. (But the proof will work for any finite $\Sigma$)

Therefore, the function mapping every TM $M$ to its language $L(M)$, *fails to cover all possible languages*