CS154, Lecture 7:
Turing Machines
Turing Machine (1936)
ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTScheidungsproblem

By A. M. Turing.

[Received 28 May, 1936.—Read 15 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 concept of a computable function of a real variable.
Great, a warehouse filled with miles and miles of rewritable tape! What are we ever going to do with this, Alan?

And thus the Turing Machine was born.
Turing Machines versus DFAs

TM can both write to and read from the tape.

The head can move left and right.

The input doesn’t have to be read entirely, and the computation can continue further (even, forever) after all input has been read.

Accept and Reject take immediate effect.
This Turing machine *decides* the language $\{0\}$.
This Turing machine *recognizes* the language \{0\}.
Deciding the language $L = \{ w\#w \mid w \in \{0,1\}^* \}$

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 with $X$, and check if the first bit $b$ to the right of the $\#$ is identical. (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

and so on...
Definition: A Turing Machine is a 7-tuple \( \Gamma = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \), where:

- \( Q \) is a finite set of states
- \( \Sigma \) is the input alphabet, where \( \Box \notin \Sigma \)
- \( \Gamma \) is the tape alphabet, where \( \Box \in \Gamma \) 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 \( q_{\text{reject}} \neq q_{\text{accept}} \)
Turing Machine Configurations

q₇

1 1 0 1 0 0 0 1 1 1 0

corresponds to the configuration:

11010q₇00110 ∈ \{Q UR\}*
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

Suppose $\delta(q_1, b) = (q_2, c, L)$ Then $aaq_1bb$ yields $aq_2acb$

Suppose $\delta(q_1, a) = (q_2, c, R)$ Then $cabq_1a$ yields $cabcq_2$

Let $w \in \Sigma^*$ and $M$ be a Turing machine $M$ accepts $w$ if there are configs $C_0, C_1, ..., C_k$, s.t.

- $C_0 = q_0w$
- $C_i$ yields $C_{i+1}$ for $i = 0, ..., k-1$, and
- $C_k$ contains the accept state $q_{\text{accept}}$
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 $L$
\[ w \in \Sigma^* \]

\[ w \in L \, ? \]

yes \rightarrow accept

no \rightarrow reject

L is **decidable** (recursive)

---

\[ w \in \Sigma^* \]

\[ w \in L \, ? \]

TM

yes \rightarrow accept

no \rightarrow reject \ or \ loop

L is **recognizable** (recursively enumerable)
A Turing machine for deciding \( \{ 0^{2^n} \mid n \geq 0 \} \)

Turing Machine PSEUDOCODE:

1. Sweep from left to right, cross 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, reject
4. Move the head back to the first input symbol.
5. Go to step 1.

Why does this work?

Idea: Every time we return to stage 1, the number of 0’s on the tape has been halved.
\{ 0^{2^n} | n \geq 0 \}

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

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

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

Step 4:
- \( x \rightarrow x, L \)
- \( 0 \rightarrow 0, L \)
\( \{ 0^{2n} \mid n \geq 0 \} \)
Multitape Turing Machines

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$
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
Theorem: $L$ is decidable if and only if both $L$ and $\overline{L}$ are recognizable.
Recall: Given \( L \subseteq \Sigma^* \), define \( \overline{L} := \Sigma^* \setminus L \)

Theorem: \( L \) is decidable iff both \( L \) and \( \overline{L} \) are recognizable

Given:
- a TM \( M_1 \) that recognizes \( L \) and
- a TM \( M_2 \) that recognizes \( \overline{L} \),
want to build a new machine \( M \) that decides \( L \)

How? Any ideas?
\( M_1 \) always accepts \( x \), when \( x \) is in \( L \)
\( M_2 \) always accepts \( x \), when \( x \) isn’t in \( L \)
Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* \setminus L$

Theorem: $L$ is decidable iff both $L$ and $\neg L$ are recognizable

Given:
- a TM $M_1$ that recognizes $L$ and
- a TM $M_2$ that recognizes $\neg L$,

want to build a new machine $M$ that decides $L$

Simulate $M_1(x)$ on one tape, $M_2(x)$ on another.
Exactly one of the two will accept
If $M_1$ accepts then accept
If $M_2$ accepts then reject
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 $N$.

Proof Idea (more details in Sipser)
Pick a natural ordering on all 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 $C_0$, $C_1$, ..., $C_k$ is some accepting computation history for $N$ on $w$. If so, accept.
It is all Zeros and Ones

One of the most popular and overemphasized clichés about computer scientists. Proxy for: genius / nerd / insensitive / anti-social / ...
(still bits are quite fundamental, “more than atoms”)

Bit Strings Encoding

Encode a finite string in $\Sigma^*$ as a bit string: encode each character as $\log |\Sigma|$ bits.

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

For $x, y \in \Sigma^*$, to encode the pair of $x$ and $y$ can add as $x, y$ over $\Sigma’ = \Sigma \cup \{,,\}$.

Or sometimes better: $(x, y) := 0|b_\Sigma(x)|1 b_\Sigma(x) b_\Sigma(y)$
TM Encoding

Can encode a TM as a bit string:

- n (states), m (tape symbols), (first) k (are input symbols),
- s (start state), t (accept state), r (reject state), u (blank symbol),
- transition1, transition2, …

( (p, i), (q, j, L) ), ( (p, i), (q, j, R) ) , …

Similarly, we can encode DFAs and NFAs as bit strings
Other ways to encode a TM exist:

\[
\begin{aligned}
\text{n states} & \quad \text{start state} & \quad \text{reject state} \\
\text{m tape symbols} & \quad \text{(first} k \text{ are input symbols)} & \quad \text{accept state} & \quad \text{blank symbol} \\
0^n10^m10^k10^s10^t10^r10^u1 & \quad & \quad \\
\end{aligned}
\]

\[
( (p, i), (q, j, L) ) = 0^p10^i10^q10^j10
\]

\[
( (p, i), (q, j, R) ) = 0^p10^i10^q10^j100
\]
Binary languages about computations

Define the following languages over \{0,1\}:

\[ A_{DFA} = \{ (B, w) \mid B \text{ encodes a DFA over some } \Sigma, \]
\[ \text{and } B \text{ accepts } w \in \Sigma^* \} \]

\[ A_{NFA} = \{ (B, w) \mid B \text{ encodes an NFA, } B \text{ accepts } w \} \]

\[ A_{TM} = \{ (M, w) \mid M \text{ encodes a TM, } M \text{ accepts } w \} \]
$A_{TM} = \{(M, w) \mid M \text{ encodes a TM over some } \Sigma, w \text{ encodes a string over } \Sigma \text{ and } M \text{ accepts } w\}$

Technical Note:

We’ll use an decoding of pairs, TMs, and strings so that every binary string decodes to some pair $(M, w)$.

If $x \in \{0,1\}^*$ doesn’t decode to $(M,w)$ in the usual way, then we define that $x$ decodes to the pair $(D, \varepsilon)$, where $D$ is a “dummy” TM that accepts nothing.

Then, we can define the complement of $A_{TM}$ very simply:

$\neg A_{TM} = \{(M, w) \mid M \text{ does not accept } w\}$
Universal Turing Machines

Theorem: There is a Turing machine $U$ which takes as input:

1. the code of an arbitrary TM $M$
2. an input string $w$

such that $U$ accepts $(M, w) \iff M$ 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: $A_{DFA}$ and $A_{NFA}$ are not regular.
The Church-Turing Thesis

Everyone’s Intuitive Notion = Turing Machines of Algorithms

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

And it has and is still been tested
\[ A_{\text{DFA}} = \{ (D, w) \mid D \text{ is a DFA that accepts string } w \} \]

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

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

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

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

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

**Theorem:** \( A_{\text{TM}} \) is recognizable but not decidable!
There are non-recognizable languages

Assuming the Church-Turing Thesis, this means there are problems that NO computing device can solve!

We can prove this using a counting argument: We will show there is no onto function from the set of all Turing Machines to the set of all languages over \( \{0,1\} \) (works for any finite \( \Sigma \))

That is, every mapping from Turing machines to languages fails to cover all possible languages
“There are more problems to solve than there are programs to solve them.”
Let $L$ be any set and $2^L$ be the power set of $L$

**Theorem:** There is no onto function from $L$ to $2^L$

**Proof:** Assume, for a contradiction, there is an onto function $f : L \rightarrow 2^L$

Define $S = \{ x \in L \mid x \notin f(x) \} \in 2^L$

If $f$ is onto, then there is a $y \in L$ with $f(y) = S$

Suppose $y \in S$. By definition of $S$, $y \notin f(y) = S$

Suppose $y \notin S$. By definition of $S$, $y \in f(y) = S$

*Contradiction!*
Let \( f : L \to 2^L \) be an arbitrary function

Define \( S = \{ x \in L \mid x \notin f(x) \} \in 2^L \)

For all \( x \in L \),

- If \( x \in S \) then \( x \notin f(x) \) [by definition of \( S \)]
- If \( x \notin S \) then \( x \in f(x) \)

In either case, we have \( f(x) \neq S \). (Why?)

Therefore \( f \) is not onto!

\[
\begin{align*}
\text{Theorem: There is no onto function from } L \text{ to } 2^L \\
\end{align*}
\]
What does this mean?

No function from $L$ to $2^L$ can “cover” all the elements in $2^L$.

No matter what the set $L$ is, the power set $2^L$ always has strictly larger cardinality than $L$. 
Thm: There are non-recognizable languages

Proof: Suppose all languages are recognizable. Then for all L, there’s a Turing machine M for recognizing L. Hence there is an onto R: \{Turing Machines\} → \{Languages\}

\{Turing Machines\} \subseteq \mathbb{M} \quad \{Languages over \{0,1\}\} \quad \{\text{Sets of strings of 0s and 1s}\} \quad \{2^\mathbb{M}\}

But there is no onto function from \{Turing Machines\} \subseteq \mathbb{M} to 2^\mathbb{M}. Contradiction!
In the early 1900’s, logicians were trying to define consistent foundations for mathematics.

Suppose $X$ = “Universe of all possible sets”

Frege’s Axiom: Let $f : X \to \{0,1\}$
Then $\{S \in X \mid f(S) = 1\}$ is a set.

Define $F = \{ S \in X \mid S \notin S \}$

Suppose $F \in F$. Then by definition, $F \notin F$.
So $F \notin F$ and by definition $F \in F$.
This logical system is inconsistent!