Today: Undecidability, Recognition, Enumeration, all that good stuff

Definition: A Turing Machine is a 7-tuple $\tau = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:
- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \notin \Gamma$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$

A TM recognizes a language if it accepts all and only those strings in the language
A language is called recognizable or recursively enumerable, (r.e.) if some TM recognizes it

A TM decides a language if it accepts all strings in the language and rejects all strings not in the language
A language is called decidable (or recursive) if some TM decides it

Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* - L$
Theorem: $L$ is decidable iff both $L$ and $\neg L$ are recognizable

Given:
a TM $M_1$ that recognizes $A$ and
a TM $M_2$ that recognizes $\neg A$,
we can build a new machine $M$ that decides $A$

Simulate $M_1(x)$ on one tape, $M_2(x)$ on another. One of them must halt. If $M_1$ halts then accept. If $M_2$ halts then reject.

There are languages over $\{0,1\}$ that are not decidable
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 $\{0,1\}$. (But it works for any $\Sigma$)

That is, every mapping from Turing machines to languages must “miss” some language.
There are languages over \( \{0,1\} \) that are not recognizable

Proof Outline:
1. Every recognizable language can be associated with a Turing machine
2. Every TM corresponds to a bit string
3. So there is a 1-1 mapping from the set of all recognizable languages, to \( \{0,1\}^* \)
4. But the set of all languages has a bijection with the POWER SET of \( \{0,1\}^* \)
5. The power set of \( A \) is always larger than \( A \), so there must be unrecognizable languages

There are undecidable (and unrecognizable) languages over \( \{0,1\} \)

\[
\begin{array}{c}
\{ \text{Recognizable languages over } \{0,1\} \} \\
\downarrow \\
\{ \text{Turing Machines} \} \\
\downarrow \\
\{0,1\}^* \\
\downarrow \\
\{ \text{Sets of strings of } 0\text{s and } 1\text{s} \} \\
\downarrow \\
\text{Set } L \\
\text{Set of all subsets of } L: 2^L
\end{array}
\]

There are (many) unrecognizable languages

"There are more problems to solve than there are programs to solve them."

Russell’s Paradox in Set Theory

In the early 1900’s, logicians were trying to define consistent foundations for mathematics.

Suppose \( X = \) “universe of all possible objects”

"Frege's Axiom" Let \( P : X \rightarrow \{0,1\} \).

Then \( \{ S \in X \mid P(S) = 1 \} \) 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 \( F \in F \).

\textit{This set theory is inconsistent!}
A Concrete Undecidable Problem

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

Theorem: $A_{tm}$ is recognizable but NOT decidable

Define a TM $U$ as follows: $U$ is a universal TM

On input $(M, w)$, $U$ runs $M$ on $w$. If $M$ ever accepts, accept. If $M$ ever rejects, reject.

Therefore, $U$ accepts $(M, w) \iff M$ accepts $w \iff (M, w) \in A_{tm}$

Therefore, $U$ recognizes $A_{tm}$

Let $\mathbb{Z}^* = \{1, 2, 3, 4, \ldots \}$.
There is a bijection between $\mathbb{Z}^*$ and $\mathbb{Z}^* \times \mathbb{Z}^*$

$(1, 4) \quad (1, 2) \quad (1, 3) \quad (1, 5) \ldots$
$(2, 4) \quad (2, 2) \quad (2, 3) \quad (2, 5) \ldots$
$(3, 4) \quad (3, 2) \quad (3, 3) \quad (3, 5) \ldots$
$(4, 4) \quad (4, 2) \quad (4, 3) \quad (4, 5) \ldots$
$(5, 4) \quad (5, 2) \quad (5, 3) \quad (5, 5) \ldots$

### OUTPUT OF $H(x, y)$

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$\ldots$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

$D$ reject reject accept accept
**OUTPUT OF D(M)**

<table>
<thead>
<tr>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>...</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td></td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

D(M) outputs the opposite of H(M,M)

**A_{TM} = \{ (M,w) | M is a TM that accepts string w \}**

A_{TM} is undecidable: (constructive proof)

Assume machine H recognizes A_{TM}

\[ H( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Rejects or loops otherwise} & \text{} 
\end{cases} \]

Construct a new TM D_H as follows: on input M, run H on (M,M) and output the “opposite” of H whenever you get an answer from H.

That is:

Given any machine H recognizing A_{TM}

(H is a potential decider for A_{TM})

we can effectively construct an instance which does not belong to A_{TM} (namely, (D_H, D_H))

but H runs forever on the input (D_H, D_H)

So H cannot decide A_{TM}

Theorem: A_{TM} is recognizable but NOT decidable

Corollary: \neg A_{TM} is not recognizable!

\neg A_{TM} = \{ (M,w) | M does not accept string w \}

Proof: Suppose \neg A_{TM} is recognizable.

Then \neg A_{TM} and A_{TM} are both recognizable...

But that would mean they’re decidable...

**The Halting Problem**

HALT_{TM} = \{ (M,w) | M is a TM that halts on string w \}

Theorem: HALT_{TM} is undecidable

Proof: Assume (for a contradiction) there is a TM H that decides HALT_{TM}

We use H to construct a TM M’ that decides A_{TM}

M'(M,w): run H(M,w)

If H rejects then reject

If H accepts, run M on w until it halts:

Accept if M accepts and

Reject if M rejects
The Emptiness Problem

\[ \text{EMPTY} = \{ M \mid M \text{ is a TM such that } L(M) = \emptyset \} \]

Given a program, does it always reject?

Theorem: \( \text{EMPTY} \) is undecidable

Proof: Assume (for a contradiction) there is a TM \( E \) that decides \( \text{EMPTY} \)

We’ll use it to get a decider \( D \) for \( A_{\text{TM}} \)

\[ D(M, w) := \text{Build a TM } M' \text{ with the behavior:} \]

\[ "M'(x) := \text{if } x \neq w \text{ reject else sim } M(w)" \]

Run \( E(M') \).

If \( E \) accepts, reject. If \( E \) rejects, accept.

The Complement of Emptiness

\[ \neg \text{EMPTY} = \{ M \mid M \text{ is a TM such that } L(M) \neq \emptyset \} \]

Given a program, does it accept some input?

Theorem: \( \neg \text{EMPTY} \) is recognizable

Proof:

\[ M'(M) := \]

For all pairs of positive integers \( i, t \),

Let \( x \) be the \( i \)th string in lexicographic order.

Determine if \( M(x) \) accepts within \( t \) steps.

If yes then accept.

L(\( M \)) \( \neq \emptyset \) if and only if \( M' \) halts and accepts \( M \)

The Emptiness Problem

Theorem: \( \text{EMPTY} \) is undecidable

Theorem: \( \neg \text{EMPTY} \) is recognizable

Corollary: \( \text{EMPTY} \) is unrecognizable

Proof: If \( \neg \text{EMPTY} \) and \( \text{EMPTY} \) recognizable, that would imply \( \text{EMPTY} \) is decidable

Mapping Reductions

\( f : \Sigma^* \rightarrow \Sigma^* \) is a computable function if there is a Turing machine \( M \) that halts with just \( f(w) \) written on its tape, for every input \( w \).

A language \( A \) is mapping reducible to language \( B \), written \( A \leq_m B \), if there is a computable \( f : \Sigma^* \rightarrow \Sigma^* \), such that for every \( w \),

\[ w \in A \iff f(w) \in B \]

\( f \) is called a mapping reduction (or many-one reduction) from \( A \) to \( B \)
Next Episode:
Undecidability and Reductions Between Problems