Undecidability

Reading: Chapter 8 & 9
Decidability vs. Undecidability

- There are two types of TMs (based on halting):
  (Recursive)
  - **TMs that always halt**, no matter accepting or non-accepting \( \equiv \) **DECIDABLE PROBLEMS**
  (Recursively enumerable)
  - **TMs that are guaranteed to halt only on acceptance**. If non-accepting, it may or may not halt (i.e., could loop forever).

- **Undecidability:**
  - Undecidable problems are those that are **not** recursive
Recursive, RE, Undecidable languages

Non-RE Languages
(all other languages for which no TMs can be built)

No TMs exist

TMs that always halt

LBA

TMs that may or may not halt

Recursive

Recursively Enumerable (RE)

“Decidable” problems

“Undecidable” problems

Regular (DFA)

Context-free (PDA)

Context sensitive

"Decidable" problems

"Undecidable" problems
Recursive Languages & Recursively Enumerable (RE) languages

- Any TM for a **Recursive** language is going to look like this:

  ![Diagram](image)

  - W → M
  - “accept”
  - “reject”

- Any TM for a **Recursively Enumerable** (RE) language is going to look like this:

  ![Diagram](image)

  - W → M
  - “accept”
Closure Properties of:
- the Recursive language class, and
- the Recursively Enumerable language class
Recursive Languages are closed under complementation

- If \( L \) is Recursive, \( \overline{L} \) is also Recursive
Are Recursively Enumerable Languages closed under complementation?  (NO)

- If $L$ is RE, $\overline{L}$ need not be RE
Recursive Langs are closed under Union

- Let $M_u = TM$ for $L_1 \cup L_2$
- $M_u$ construction:
  1. Make 2-tapes and copy input $w$ on both tapes
  2. Simulate $M_1$ on tape 1
  3. Simulate $M_2$ on tape 2
  4. If either $M_1$ or $M_2$ accepts, then $M_u$ accepts
  5. Otherwise, $M_u$ rejects.
Recursive Langs are closed under Intersection

- Let $M_n = TM$ for $L_1 \cap L_2$
- $M_n$ construction:
  1. Make 2-tapes and copy input $w$ on both tapes
  2. Simulate $M_1$ on tape 1
  3. Simulate $M_2$ on tape 2
  4. If $M_1$ AND $M_2$ accepts, then $M_n$ accepts
  5. Otherwise, $M_n$ rejects.
Other Closure Property Results

- Recursive languages are also closed under:
  - Concatenation
  - Kleene closure (star operator)
  - Homomorphism, and inverse homomorphism

- RE languages are closed under:
  - Union, intersection, concatenation, Kleene closure

- RE languages are *not* closed under:
  - Complementation
“Languages” vs. “Problems”

A “language” is a set of strings

Any “problem” can be expressed as a set of all strings that are of the form:

- “<input, output>”

E.g., Problem (a+b) ≡ Language of strings of the form { “a#b, a+b” }

==> Every problem also corresponds to a language!!

Think of the language for a “problem” == a verifier for the problem
The Halting Problem

An example of a recursive enumerable problem that is also undecidable
The Halting Problem

Non-RE Languages

Recursive

Context sensitive

Regular (DFA)

Context-free (PDA)
What is the Halting Problem?

Definition of the “halting problem”:

- Does a given Turing Machine $M$ halt on a given input $w$?
The Universal Turing Machine

- **Given:** TM $M$ & its input $w$
- **Aim:** Build another TM called “$H$”, that will output:
  - “accept” if $M$ accepts $w$, and
  - “reject” otherwise

- An algorithm for $H$:
  - Simulate $M$ on $w$
  - $H(<M,w>) = \begin{cases} 
    \text{accept,} & \text{if } M \text{ accepts } w \\
    \text{reject,} & \text{if } M \text{ does not accept } w 
  \end{cases}$

**Implies:** $H$ is in RE

**Question:** If $M$ does *not* halt on $w$, what will happen to $H$?
A Claim

- **Claim**: No H that is always guaranteed to halt, can exist!

- **Proof** (Alan Turing, 1936)
  - By contradiction, let us assume H exists

![Diagram](Image)
Therefore, if H exists \( \Rightarrow \) D also should exist. But can such a D exist? (if not, then H also cannot exist)

**HP Proof (step 1)**

- Let us construct a new TM D using H as a subroutine:
  - On input \(<M>:\):
    1. Run H on input \(<M, <M>>; \) // (i.e., run M on M itself)
    2. Output the *opposite* of what H outputs;
The notion of inputing “<M>” to M itself

A program can be input to itself (e.g., a compiler is a program that takes any program as input)

\[ D (<M>) = \begin{cases} 
  \text{accept, if } M \text{ does not accept } <M> \\
  \text{reject, if } M \text{ accepts } <M> 
\end{cases} \]

Now, what happens if D is input to itself?

\[ D (<D>) = \begin{cases} 
  \text{accept, if } D \text{ does not accept } <D> \\
  \text{reject, if } D \text{ accepts } <D> 
\end{cases} \]

A contradiction!!! \( \Rightarrow \) Neither D nor H can exist.
Of Paradoxes & Strange Loops

E.g., Barber’s paradox, Achilles & the Tortoise (Zeno’s paradox)
MC Escher’s paintings

A fun book for further reading:

“Godel, Escher, Bach: An Eternal Golden Braid”
by Douglas Hofstadter (Pulitzer winner, 1980)
The Diagonalization Language

Example of a language that is not recursive enumerable

(i.e, no TMs exist)
The Halting Problem

The Diagonalization language

Non-RE Languages

Regular (DFA)

Context-free (PDA)

Context sensitive

Recursive

Recursively Enumerable (RE)
A Language about TMs & acceptance

Let L be the language of all strings <M,w> s.t.:
1. M is a TM (coded in binary) with input alphabet also binary
2. w is a binary string
3. M accepts input w.
Enumerating all binary strings

- Let w be a binary string
- Then $1w \equiv i$, where i is some integer
  - E.g., If $w=\varepsilon$, then $i=1$;
  - If $w=0$, then $i=2$;
  - If $w=1$, then $i=3$; so on...
- If $1w \equiv i$, then call w as the $i^{th}$ word or $i^{th}$ binary string, denoted by $w_i$.
- $\implies$ A *canonical ordering* of all binary strings:
  - $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 100, 101, 110, \ldots\}$
  - $\{w_1, w_2, w_3, w_4, \ldots, w_i, \ldots\}$
Any TM $M$ can also be binary-coded

- $M = \{ Q, \{0,1\}, \Gamma, \delta, q_0, B, F \}$

- Map all states, tape symbols and transitions to integers (==>binary strings)
- $\delta(q_i, X_j) = (q_k, X_l, D_m)$ will be represented as:
  - $\Rightarrow 0^i10^j10^k10^l10^m$

**Result:** Each TM can be written down as a long binary string

$\Rightarrow$ Canonical ordering of TMs:

- $\{M_1, M_2, M_3, M_4, \ldots M_i, \ldots \}$
The Diagonalization Language

- \( L_d = \{ w_i | w_i \not\in L(M_i) \} \)
  - The language of all strings whose corresponding machine does not accept itself (i.e., its own code)

<table>
<thead>
<tr>
<th>(input word w)</th>
<th>1 2 3 4 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>1 0 1 0 1 ...</td>
</tr>
<tr>
<td>(TMs) i</td>
<td>2 1 1 0 0 ...</td>
</tr>
</tbody>
</table>

- Table: \( T[i,j] = 1 \), if \( M_i \) accepts \( w_j \) = 0, otherwise.

- Make a new language called \( L_d = \{ w_i | T[i,i] = 0 \} \)
\[ L_d \text{ is not RE (i.e., has no TM)} \]

- **Proof (by contradiction):**
  - Let \( M \) be the TM for \( L_d \)
  - \( \implies M \) has to be equal to some \( M_k \) s.t. \( L(M_k) = L_d \)
  - \( \implies \) Will \( w_k \) belong to \( L(M_k) \) or not?
    1. If \( w_k \in L(M_k) \implies T[k,k]=1 \implies w_k \notin L_d \)
    2. If \( w_k \notin L(M_k) \implies T[k,k]=0 \implies w_k \in L_d \)
  - A contradiction either way!!
Why should there be languages that do not have TMs?

We thought TMs can solve everything!!
Non-RE languages

How come there are languages here?
(e.g., diagonalization language)

Non-RE Languages

Regular (DFA)
Context-free (PDA)
Context-sensitive
Recursive
Recursively Enumerable (RE)
One Explanation

There are more languages than TMs

- By pigeon hole principle:
  - ==> some languages cannot have TMs

- But how do we show this?

- Need a way to “count & compare” two infinite sets (languages and TMs)
How to count elements in a set?

Let A be a set:

- If A is finite ==> counting is trivial
- If A is infinite ==> how do we count?
- And, how do we compare two infinite sets by their size?
Cantor’s definition of set “size” for infinite sets (1873 A.D.)

Let \( N = \{1,2,3,\ldots\} \) (all natural numbers)
Let \( E = \{2,4,6,\ldots\} \) (all even numbers)

Q) Which is bigger?

A) Both sets are of the same size
   - “Countably infinite”
   - Proof: Show by one-to-one, onto set correspondence from \( N \implies E \)

\[
\begin{array}{c|c}
 n & f(n) \\
\hline
 1 & 2 \\
 2 & 4 \\
 3 & 6 \\
 \cdot & \cdot \\
\end{array}
\]

i.e, for every element in \( N \), there is a unique element in \( E \), and vice versa.
Example #2

- Let Q be the set of all rational numbers
- \[ Q = \{ \frac{m}{n} \mid \text{ for all } m, n \in \mathbb{N} \} \]
- **Claim:** Q is also countably infinite; \( \implies |Q| = |\mathbb{N}| \)
Really, really big sets!
(even bigger than countably infinite sets)

Uncountable sets

Example:
- Let $\mathbb{R}$ be the set of all real numbers
- **Claim:** $\mathbb{R}$ is uncountable

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
<th>Build $x$ s.t. $x$ cannot possibly occur in the table</th>
<th>E.g. $x = 0.2644...$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$3.14159...$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$5.555...$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$0.12345...$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$0.51430...$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Therefore, some languages cannot have TMs…

- The set of all TMs is countably infinite

- The set of all Languages is uncountable

===> There should be some languages without TMs (by PHP)
The problem reduction technique, and reusing other constructions
Languages that we know about

- **Language of a Universal TM ("UTM")**
  - \( L_u = \{ <M,w> \mid M \text{ accepts } w \} \)
  - **Result:** \( L_u \) is in RE but not recursive

- **Diagonalization language**
  - \( L_d = \{ w_i \mid M_i \text{ does not accept } w_i \} \)
  - **Result:** \( L_d \) is non-RE
TMs that accept non-empty languages

- \( L_{ne} = \{ M \mid L(M) \neq \emptyset \} \)
- \( L_{ne} \) is RE
- **Proof:** (build a TM for \( L_{ne} \) using UTM)

Non-deterministic Simulator for \( L_{ne} \)

\[ \text{Guess } w \]

\[ \text{UTM} \]

\[ \text{"accept"} \]
TMs that accept non-empty languages

- $L_{ne}$ is not recursive
- **Proof:** ("Reduce" $L_u$ to $L_{ne}$)
  - Idea: $M$ accepts $w$ if and only if $L(M') \neq \emptyset$
Reducability

- **To prove:** Problem $P_1$ is undecidable
- **Given/known:** Problem $P_2$ is undecidable
- **Reduction idea:**
  1. “Reduce” $P_2$ to $P_1$:
     - Convert $P_2$’s input instance to $P_1$’s input instance s.t.
       1) $P_2$ decides only if $P_1$ decides
  2. Therefore, $P_2$ is decidable
  3. A contradiction
  4. Therefore, $P_1$ has to be undecidable
The Reduction Technique

Reduce $P_2$ to $P_1$:

- **Construct**
  - $P_2$ instance
  - $P_1$ instance

- **Decide**
  - yes
  - no

**Conclusion**: If we could solve $P_1$, then we can solve $P_2$ as well
Summary

- Problems vs. languages
- Decidability
  - Recursive
- Undecidability
  - Recursively Enumerable
  - Not RE
  - Examples of languages
- The diagonalization technique
- Reducibility