Decidability

Previously:
- Turing machines as a model of what is computable
- Introduce problems that cannot be computed

Today:
- (Un)decidability
- Diagonalization proofs

Procedures Versus Algorithms

There are two senses in which a TM accepts a language:
1. The TM accepts the strings in the language (by final state), but does not halt on some of the strings not in the language:
   - Thus, we can never be sure whether those strings are rejected, or eventually will be accepted
   - A language accepted in this way is called recursively enumerable (RE)
   - Such a language is called undecidable
   - The TM is sometimes referred to as a procedure
2. The TM accepts by final state, but halts on every string, whether or not it is accepted:
   - A language accepted this way is called recursive
   - As a problem, the question is called decidable
   - The TM is called an algorithm
Decidable Problems

- Acceptance problem for DFAs
  - Does a particular DFA accept a given string
  - Can represent this problem as a language
    \[ A_{\text{DFA}} = \{ <B,w> \mid B \text{ is a DFA that accepts } w \} \]
- Problem of testing whether DFA \( B \) accepts \( w \)
  same as testing if \( <B,w> \) is a member of \( A_{\text{DFA}} \)
- Prove by designing a TM that decides \( A_{\text{DFA}} \)

Proof

- Proof given here will be in high-level descriptive language (like pseudocode), rather than explicitly drawing out state diagrams
- Write the proof in quotes to emphasize that the description is informal but there is a precise mathematical formulation

\( A_{\text{DFA}} \) is Decidable

\[ M = \text{“on input } <B,w>, \text{ where } B \text{ is a DFA and } w \text{ is a string:} \]
1. Simulate \( B \) on \( w \)
2. If the simulation ends in an accept state, \( \text{accept} \). If it ends in a non-accepting state, \( \text{reject} \).”

Other decidable problems

- Emptiness Problem for DFAs
  \[ E_{\text{DFA}} = \{ <A> \mid A \text{ is a DFA and } L(A) \text{ is empty} \} \]

\[ M = \text{“on input } <A>, \text{ where } A \text{ is a DFA:} \]
1. Mark the start state of \( A \)
2. Repeat until no new states get marked:
   - Mark any state that has a transition coming into it from any state that is already marked
3. If no accept state is marked, \( \text{accept} \). Otherwise, \( \text{reject} \).”
Other decidable problems

Equivalence Problem for DFAs

\[ EQ_{\text{DFA}} = \{ <A,B> | A, B \text{ are DFAs and } L(A) = L(B) \} \]

\( F = \) “on input \(<A,B>\), where \( A \) and \( B \) are DFAs:
1. Construct DFA \( C \) to accept strings that are accepted by either \( A \) or \( B \) but not both
2. Run \( E_{\text{DFA}} \) on input \(<C>\)
3. If \( E_{\text{DFA}} \) accepts, accept. If \( E_{\text{DFA}} \) rejects, reject.”

Parsing problem

\[ A_{\text{CFG}} = \{ <A,w> | A \text{ is a CFG and } w \text{ is in } L(A) \} \]

\( S = \) “on input \(<G,w>\), where \( G \) is a CFG and \( w \) is a string:
1. Convert \( G \) to an equivalent grammar in Chomsky normal form
2. List all derivations with \( 2n-1 \) steps, where \( n \) is the length of \( w \)
3. If any of these derivations generate \( w \), accept. Otherwise, reject.”

Undecidable problems

• **Undecidable** problems have no algorithm, regardless of whether or not they are accepted by a TM that fails to halt on some inputs
• We will prove undecidable the following problem:
  
  Does this TM accept this input?

An undecidable problem

\[ A_{\text{TM}} = \{ <M,w> | M \text{ is a TM and } M \text{ accepts } w \} \]

• But it is recursively enumerable (“Turing-recognizable”)
  – So recognizers are more powerful than deciders!

\( U = \) “On input \(<M,w>\) where \( M \) is a TM and \( w \) is a string
1. Simulate \( M \) on input \( w \)
2. If \( M \) ever enters an accept state, accept; if \( M \) ever enters a reject state, reject”
• But \( U \) loops on input if \( M \) loops on \( w \), so does not **decide** \( A_{\text{TM}} \)
Goal

Prove undecidable the language consisting of pairs $<M, w>$ such that
1. $M$ is a TM with input alphabet $\{1, 0\}$
2. $w$ is a string of 0s and 1s
3. $M$ accepts input $w$

$A_{TM} = \{<M, w> \mid M$ is a TM and $M$ accepts $w\}$

If this problem with restricted inputs is undecidable, then the more general problem where TMs may have any alphabet is surely undecidable

Plan

1. **Show a particular language not to be RE**
   - Like the “hello-world” argument, we show no TM can tell whether a given TM halts on a given input – the proof is by “diagonalization” or self-reference
2. **Use the non-RE language from (1) to show another language to be RE, but not recursive**
   - **Trick:** if a language and its complement are both RE, then they are both recursive
     - Can you figure out why?
   - Thus, if a language $L$ is RE, but its complement is not, then $L$ is not recursive
3. **Use this method to show that the Halting Problem is undecidable**
   - Halting Problem is “does a given TM $M$ halt given input string $w$, regardless of whether it accepts or rejects?”

The Diagonalization Method

- Use to prove the undecidability of the halting problem
- Discovered by Cantor in 1873 in the process of trying to determine how to tell which of two infinite sets is larger
  - Cantor observed that two finite sets have the same size if the elements of one can be paired with the other
  - Extended this to infinite sets
Encoding TMs as Integers

We will focus on TMs whose input alphabet is \{0, 1\}. Each such TM can be represented by one or more integers, using the following code:

- Assume the **states** are \{q_1, q_2, \ldots\}. **Represent** \( q_i \) by \( 0^i \)
- Assume the **tape symbols** are \{X_1, X_2, \ldots\}, where the first three of these are 0, 1, and \( B \), in that order. **Represent** \( X_i \) by \( 0^i \)
- **Represent** directions \( L \) and \( R \) by 0 and 00, respectively, and refer to them as \( L = D_1, R = D_2 \)
- **Represent** a **rule** of the TM \( \delta(q_i, X_j) = (q_k, X_l, D_m) \) by \( 0^i10^j10^k10^l10^m \)

- **Represent** the whole TM by \( 111 C_111 C_211 \ldots 11C_n111 \), where \( C_i \) is the code for one of the \( \delta \) rules, in any order
- **Conversely**, every integer \( i \) can be said to describe some TM \( M_i \)

  - If \( i \) in binary is not of the right form (111 code...), then \( M_i \) is the TM with no moves
  - Note that many integers represent the same TM, but that is neither good nor bad

Facts about TMs

- **The set of all TMs is infinite**
  - Any TM has a finite encoding as a string over \{0, 1\}.
  - There is an infinite number of such strings
- **The set of all TMs is countable**
  - Each TM corresponds to an integer interpretation of the string of 0s and 1s used to represent it
  - We call a set **countable** if either it is finite or it has the same size as \( \mathbb{N} \), the set of natural numbers \( \{1, 2, 3, \ldots\} \)
- **The set of all TMs is Recursively Enumerabile**
  - Any countable set can be produced by an **enumeration procedure** – i.e., a method by which its elements can be written in some sequence

The Diagonalization Language

- Define \( L_d \) to be the **set of binary strings** \( w \) with the following properties:
  1. First, let \( i \) be the integer that is \( w \) in binary
    - Refer to \( w \) as the “ith string” or \( w_i \)
  2. \( w_i \) is in \( L_d \) if and only if \( w_i \) is not in \( L(M_i) \)
    - That is, \( L_d \) consists of all strings \( w \) such that the TM \( M \) whose code is \( w \) does not accept when given \( w \) as input
Suppose the set of all TMs is infinite.

**Proof**

- Each row is the characteristic vector for $M_i$.
- $L_d$ cannot be the characteristic vector of any TM because it disagrees in some column with every row of the table.
- Thus, the diagonalization cannot represent the language of any TM.

Use diagonalization when you want to construct an element that is different from every element on a given list. This is used in proofs by contradiction, for example, when you want to show a function can’t hit every element of a set.

---

**The Trick**

- Each row is the characteristic vector for $M_i$.
- $L_d$ cannot be the characteristic vector of any TM because it disagrees in some column with every row of the table.
- Thus, the diagonalization cannot represent the language of any TM.

---

**Another Example**

This one is sometimes easier to grasp:

- If $S$ is an infinite countable set, then its power set, $2^S$, is not countable, and hence not RE.
- **Proof:** Let $S = \{s_1, s_2, s_3, \ldots\}$. Any element $t$ of $2^S$ can be represented by a sequence of 0s and 1s with 1 in position $i$ if and only if $s_i$ is in $t$.
  - E.g., the set $\{s_2, s_3, s_6\}$ is represented by 01100100..., $\{s_1, s_3, s_5\}$ is 10101...
  - Every such sequence represents a unique element of $2^S$. 

---

**Proof $L_d$ is not recursively enumerable**

No TM exists for $L_d$.

- Suppose $L_d$ is RE. Then $L_d = L(M)$ for some TM $M$.
- Input alphabet of $M$ is $\{0, 1\}$.
- $M$ is $M_i$ for at least one value of $i$.
- Question: is $w_i$ in $L_d$?
  - **Suppose so.** Then $M_i$ accepts $w_i$ because $L_d = L(M_i)$. But by definition of $L_d$, $w_i$ is not in $L_d$ because it contains only those $w_j$ not accepted by any $M_j$ (Contradiction).
  - **Suppose not.** Then $w_i$ is in $L(M_i)$ by definition of $L_d$. But $L(M_i) = L(M) = L_d$, so $w_i$ is in $L_d$ (Contradiction).
  - Since we derive a contradiction in either case, we conclude that our assumption $L(M) = L_d$ was wrong, and in fact, there is no such TM $M$. 

---

**Diagonalization**

Each cell tells whether machine $i$ accepts string $j$.

Diagonal values tell whether machine $M_j$ accepts $w_i$.

$L_d$ is constructed by complementing the diagonal.

$L_d = 1, 0, 0, 0, \ldots$
• Suppose $2^S$ is countable; its elements can be written as $t_1, t_2, t_3...$

• Enter in a table with rows labels $t_i$, take main diagonal’s complement

|   | $t_1$ | $t_2$ | $t_3$ | $t_4$ | ...
|---|---|---|---|---|---
| $t_1$ | 1 | 0 | 0 | 0 | ...
| $t_2$ | 1 | 1 | 0 | 0 | ...
| $t_3$ | 1 | 1 | 0 | 1 | ...
| $t_4$ | 1 | 1 | 0 | 1 | ...

Complement of diagonal =

0 0 1 1

Can’t be $t_1$

Can’t be $t_2$

Can’t be $t_3$

Can’t be $t_4$

Consequence

There are fewer Turing machines than there are languages

Therefore, some languages must not be recursively enumerable

Language relationships

Recursive = decidable = algorithm = TM that always halts
Recursively enumerable = undecidable = TM accepts but does not halt
Non-RE = undecidable = no TM exists for this language

We care primarily about recursive vs. the rest – i.e., what is decidable and what is not