This lecture covers Chapter 9 of HMU: Decidability and Undecidability

- Preliminaries
- Example of a non-RE language
- Recursive languages
- Universal Language
- Reductions of Problems
- Rice’s Theorem
- Post’s Correspondence Problem
- Undecidable Problems about CFGs

Additional Reading: Chapter 9 of HMU.
Preliminaries
We can construct a bijective map $\phi$ from the set of binary strings $\{0, 1\}^*$ to Natural numbers $\mathbb{N}$.

Enlist all strings ordered by length, and for each length by lexicographic ordering.

The set of finite binary strings is countable/denumerable.
The Set of Turing Machines

> For simplicity, let’s assume that input alphabet to be binary.

> WLOG, we can assume that TMs halt at the final state. Consequently, we only need **one** final state (perhaps after collapsing all states into one).

> Consider $M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$.
  > Rename states $\{q_1, \ldots, q_k\}$ for some $k \in \mathbb{N}$.
  > Rename input alphabet using $X_1 = 0$, $X_2 = 1$, and blank $B$ as $X_3$.
  > Rename the rest of the tape symbols by $X_4, \ldots, X_\ell$ for some $\ell \in \mathbb{N}$.
  > Rename $L$ as $D_1$ and $R$ and $D_2$.

> Every transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ can be represented as a tuple $(i, j, k, l, m)$.

> Map each transition tuple $(i, j, k, l, m)$ to a **unique** binary string $0^i10^j10^k10^l10^m$. NB: No string representing a transition tuple contains 11.

> Order transition tuples lexicographically and concatenate all transitions using 11 to indicate end of a transition. Let the resultant string be $w_M$. For example, 3 transitions can be combined as

$$w_M := \underbrace{0^i10^i10^k10^l10^m}_1 \underbrace{10^i10^j10^k10^l10^m}_2 \underbrace{10^i10^j10^k10^l10^m}_3$$

1st transition 2nd transition 3rd transition

> Define the code for TM $M$ as $w_M$ and its number $\langle M \rangle := \phi(w_M)$ for each $M$.
The Set of Turing Machines

An Example

\[
\begin{align*}
q_1 & \quad 01010101001 \\
& \quad (1, 1, 1, 1, 2) \\
& \quad X_1/X_1, D_2 \\
q_2 & \quad 01001001001001 \\
& \quad (1, 2, 2, 2, 2) \\
& \quad X_2/X_2, D_2 \\
q_3 & \quad 00101001001 \\
& \quad (2, 1, 2, 1, 2) \\
& \quad X_1/X_1, D_2 \\
q_4 & \quad X_2/X_2, D_2 \\
& \quad (2, 2, 1, 2, 2) \\
& \quad 00100101001001 \\
q_5 & \quad X_3, X_3, D_1 \\
& \quad (2, 3, 3, 3, 1) \\
& \quad 00100010001000101 \\
\end{align*}
\]

\[< M >= \phi(010101010^2111010^210^210^21110^21010^21010^2111 \\
0^210^21010^210^21110^210^310^310^3101).\]

Remark 7.1.1

\> Each natural number corresponds to at most one TM.
\> There are multiple numbers that represent the ‘same’ TM.
\> The set of TMs is countable.
\> The set of RE languages is countable; so are CFLs and regular languages.
Example of a non-RE language
Diagonalization Language $L_d$

- Let $M_i$ be the TM such that $< M_i > = i$. (If for an $i$, no such TM exists, we let $M_i$ to be the TM with 1 state, no transitions and no final state, i.e., it accepts no input).
- Construct an infinite table of 0s and 1s with a 1 at the $i^{th}$ row and $j^{th}$ column if $M_i$ accepts $w_j = \phi^{-1}(j)$ (see Slide 3 for $\phi$).
- Define a language $L_d = \{ \phi^{-1}(j) : M_j$ does not accept $\phi^{-1}(j)$, where $j \in \mathbb{N} \}$.

$L_d = \{ \epsilon, 00, 10, \ldots \}$

† Entries are for illustrative purposes only
**Example of a non-RE language**

$L_d$ is not recursively enumerable language

> $L_d$ cannot be accepted by any TM.

> For each $i \in \mathbb{N}$, the string $\phi^{-1}(i)$ is exclusively in either $L_d$ or $L(M_i)$. Hence $L_d \neq L(M_i)$ for any $i \in \mathbb{N}$.

\[
\begin{array}{cccccccc}
& \epsilon & 0 & 1 & 00 & 01 & 10 & 11 & \ldots \\
\phi^{-1}(1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\phi^{-1}(2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\phi^{-1}(3) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\phi^{-1}(4) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\phi^{-1}(5) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\phi^{-1}(6) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\phi^{-1}(7) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\end{array}
\]

$L_d = \{\epsilon, 00, 10, \ldots\}$

† Entries are for illustrative purposes only
Recursive languages
Recursive Languages

- A language $L$ is **recursive** if it is accepted by a TM that halts on all inputs.
  - Every recursive language is recursively enumerable (by definition).

- A (decision) problem that is equivalent to: “is a given $w$ in a given recursive language $L$?” is said to be **decidable** (for the TM that accepts/rejects $L$ is effectively the machine description of an algorithm for solving the problem).
Recursive languages

(Some Obvious) Properties of Recursive Languages

**Theorem 7.3.1**

*If $L$ is recursive, so is $L^c$.***

**Proof of Theorem 7.3.1**

- Accepting states of $M$ are non-accepting states of $M'$.
- Add a new and only final state $q_f$ in $M'$ such that
  
  $\delta_M(q, X)$ undefined and $q \notin F$

  $\Downarrow$

  $\delta_{M'}(q, X) = (q_f, X, R)$.

- Recursive languages are closed under complementation.
(Some Obvious) Properties of Recursive Languages

**Theorem 7.3.2**

*If* $L$ *and* $L^c$ *are both recursively enumerable, then* $L$ *(and* $L^c$*) are recursive.*

**Proof of Theorem 7.3.2**

1. Let $L = L(M)$ and $L^c = L(M')$. Run $M$ and $M'$ in parallel using a 2-tape TM.
2. Both TMs cannot halt in final states, and both TMs cannot halt in non-final states.
3. Continue running both TMs until either halts in a final state. If one TM halts in a non-final state, continue until the other halts in a final state (note: the other TM will halt in a final state!).
4. Accept (or reject) if $M$ (or $M'$) halts in a final state, respectively.

**Alternate Definition of Recursive Languages**

$L$ is recursive if $L$ and $L^c$ are both recursively enumerable.
The Universal Language and Turing Machine
The Universal Language and Turing Machine

Universal Language $L_u$

$L_u := \{\langle M \rangle 111 w : \text{TM } M \text{ and } w \in L(M)\}$. [See Slide 3]

Universal TM $U$ (modelled as 5-tape TM)

1. $U$ copies $\langle M \rangle$ to tape 2 and verifies it for valid structure.
2. Copies $w$ onto tape 3 (maps $0 \mapsto 01$, $1 \mapsto 001$)
3. Initiates 4th tape with $0^1$ ($M$ starts in $q_1$)
4. To simulate a move of $M$, $U$ reads tapes 3 and 4 to identify $M$'s state and input as $0^i$ and $0^j$; if state is accepting, $M$ and hence $U$ accept their inputs and halt. Else, $U$ scans for the string $110^i10^j1$.
   > If found, using the transition, tapes 4 and 3 are updated, and tape 3's head moves to right or left.
   > If not, $M$ halts, and so does $U$. Why is scratch tape needed?
Where does $L_u$ Lie in the Hierarchy of Languages

**Theorem 7.4.1**

$L_u$ is recursively enumerable, but is not recursive.

**Proof of Theorem 7.4.1**

- $L_u$ is recursively enumerable because TM $U$ accepts it.
- Suppose it were recursive. Then, $L^c_u$ is also recursive.
- Let TM $M'$ accepts $w \in L^c_u$ and reject $w \in L^c_u$.
- Construct a TM $M''$ such that it first takes its input $w$ appends it with $111w$. It then moves to the beginning of the first $w$ and simulates $M'$.
- $M''$ accepts $w \iff w111w \in L^c_u \iff w111w \notin L_u \iff M_{\phi^{-1}(w)}$ rejects $w \iff w \in L_d$.
- Then, $L(M'') = L_d$, which is impossible!
Reductions of Problems
What is a Reduction?

- A decision problem $P$ is said to reduce to decision problem $Q$ if every instance of $P$ can be transformed to some instance of $Q$ and a yes (or no) answer to that instance of $Q$ yields a yes (or no) answer to original instance of $P$, respectively.
- Here, transform implies the existence of a Turing machine that takes an instance of $P$ written on a tape and halts with an instance of $Q$ written on it.
- Note that for deciding all instances of $P$, it is not necessary for all instances of $Q$ to be (re)solved.

Theorem 7.5.1

If a problem $P$ reduces to a problem $Q$ then:

(a) $P$ is undecidable $\Rightarrow$ $Q$ is undecidable

(b) $P$ is non-R.E. $\Rightarrow$ $Q$ is non-R.E.
Problem Reduction

Proof of Theorem 7.5.1

(a) Suppose $P$ is undecidable and $Q$ is decidable. Let TM $M_Q$ decide $Q$.
   - Consider the TM $M_P$ that first operates as TM $M_{P \rightarrow Q}$ that transforms $P$ to $Q$, and then operates as $M_Q$.
   - This is a TM that decides all instances of $P$, a contradiction.

(b) Suppose $P$ is non-R.E. and $Q$ is R.E. Then there must be a TM $M_Q$ that accepts inputs when they correspond to instances of $Q$ whose answer is yes.
   - Consider the TM $M_P$ that first operates as TM $M_{P \rightarrow Q}$, and then operates as $M_Q$.
   - Note that $M_P$ might not halt, since $M_Q$ might not.

   - This is a TM that accepts all instances of $P$ whose answer is a yes, a contradiction.
Rice's Theorem
Some More Abstract Languages

Language of TMs Accepting Empty and Non-empty Languages

> \( L_e = \{ \langle M \rangle : L(M) = \emptyset \} \).

> \( L_{ne} = L_e^c = \{ \langle M \rangle : L(M) \neq \emptyset \} \).

Theorem 7.6.1

\( L_{ne} \) is R.E.
Rice’s Theorem

**Proof of Theorem 7.6.1**

- In cycle $k$, $M'$ runs one move of $M$ for each ID, and adds the initial ID of $M$ when $\phi^{-1}(k)$ is on the tape.
- $\text{ID}(i,j)$ = the ID after $j - 1$ moves when $M$ reads $\phi^{-1}(j)$ on its tape.
- If any ID contains an accepting state, $M'$ halts as $M$ would have on that input.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Tape 1</th>
<th>Tape 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$\text{ID}(1,1)$</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>$\text{ID}(1,2) \uparrow \text{ID}(2,1)$</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>$\text{ID}(1,3) \uparrow \text{ID}(2,2) \uparrow \text{ID}(3,1)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$k$</td>
<td>101...0</td>
<td>$\text{ID}(1,k) \uparrow \text{ID}(2,k-1) \uparrow \text{ID}(3,k-2) \uparrow ... \uparrow \text{ID}(k,1)$</td>
</tr>
</tbody>
</table>
Proof of Theorem 7.6.2

> There is a TM $M_{M,w}$ that ignores its input and constantly tests if $M$ accepts $w$.
  > $M_1$ will first erase its input tape, and paste $w$ and run as $M$.

\[
\begin{array}{ccccc}
  & M_{M,w} & \quad & w & \quad & M \\
  x & \rightarrow & \quad & \rightarrow & \quad & \rightarrow \\
\end{array}
\]

> **Mind-bending step:** There is a TM $M_1$ that takes $\langle M \rangle 111w$ and outputs $\langle M_{M,w} \rangle$.
  Note: $M_1$ **always** halts (even if $M$ does not halt when input is $w$!)

\[
\begin{array}{ccc}
  \langle M \rangle 111w & \quad & M_1 \\
  \rightarrow & \rightarrow & \rightarrow \\
\end{array}
\]

> $M$ accepts $w \iff M_{m,w}$ accepts all inputs $\iff \langle M_{M,w} \rangle \in L_{ne}$

> Suppose that $L_{ne}$ is recursive. Then there is TM $M_2$ that accepts iff input $\langle M \rangle \in L_{ne}$.

> Let TM $M_3$ read $\langle M \rangle 111w$ and operate as $M_1$ and then when $M_1$ halts, operate as $M_2$. Then, $M_3$ accepts/rejects $\langle M \rangle 111w$ iff $M$ accepts/rejects $w$.

> $L_u$ is then recursive, which is a contradiction.
Rice’s Theorem

A property of RE languages is a collection (set) of RE languages over $\Sigma$.

A property is trivial if the collection it defines is empty or is all of RE languages; the property is deemed non-trivial otherwise.

Mind-bending idea: Every property can also be viewed as a language over $\Sigma$.

Let $\mathcal{P}$ be a property of languages. For every $L \in \mathcal{P}$, there exists an $M$ such that $L = L(M)$. Hence $\mathcal{P} \equiv L_{\mathcal{P}} := \{\langle M \rangle : L(M) \in \mathcal{P}\}$.

Theorem 7.6.3

Every non-trivial property $\mathcal{P}$ of RE languages is undecidable, i.e., $L_{\mathcal{P}}$ is not recursive.
Rice’s Theorem

Proof of Theorem 7.6.3

> WLOG, we can assume that \( \emptyset \notin \mathcal{P} \). Else consider \( \mathcal{P}^c \).

> Since \( \mathcal{P} \) is non-trivial, there is a language \( L \in \mathcal{P} \) and a TM \( M_L \) that accepts \( L \).

> Let \( M_{M,w} \) be a TM that runs \( M \) on \( w \) and if \( M \) accepts \( w \), then reads its input and operates as \( M_L \).

> **Mind-bending step:** There is a TM \( M_1 \) that takes \( \langle M \rangle \) and \( w \) and outputs \( \langle M_{M,w} \rangle \). Note: \( M_1 \) always halts (even if \( M \) does not halt when input is \( w \)!)  

\[
\langle M \rangle \quad w \quad \xrightarrow{\text{M}} \quad \langle M_{M,w} \rangle
\]

> \( M \) accepts \( w \) \iff \( L(M_{M,w}) = L \in \mathcal{P} \).

> If \( \mathcal{P} \) were decidable, then there is a ML \( M_2 \) such that \( M_2 \) accepts \( \langle M \rangle \) iff \( L(M) \in \mathcal{P} \).

> Then, we can devise a TM \( M_3 \) such that it reads \( \langle M \rangle \) and \( w \) operates first as \( M_1 \) and then when \( M_1 \) has halted, it operates as \( M_2 \).

> \( M_3 \) accepts/rejects \( \langle M \rangle \) \iff \( L(M_{M,w}) \in \mathcal{P} \iff M \) accepts/rejects \( w \).

> Then, \( \mathcal{L}_u \) is recursive, a contradiction.
Post’s Correspondence Problem
PCP: Definition

- Suppose we are given two ordered lists of strings over $\Sigma$, say $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$.
- We say $(u_i, v_i)$ to be a corresponding pair.
- PCP Problem: Is there a sequence of integers $i_1, \ldots, i_m$ such that $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$?
  - $m$ can be greater than $k$, the list length.
  - We can reuse pairs as many times as we like.

A PCP example

- A solution cannot start with $i_1 = 3$.
- A solution can start with $i_1 = 1$, but then $i_2 = 1$, and $i_3 = 1$. . . . Consequently, $i_1$ cannot equal 1.
- A solution does exist: $(i_1, i_2, i_3) = (2, 3, 1)$.
- $(i_1, i_2, i_3, i_4, i_5, i_6) = (2, 3, 1, 2, 3, 1)$ is also solution.
Modified PCP (MPCP): Definition

> Suppose we are given two ordered lists of strings over $\Sigma$, say $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$.

> MPCP Problem: Is there a sequence of integers $i_1, \ldots, i_m$ such that $u_1 u_{i_1} \cdots u_{i_m} = v_1 v_{i_1} \cdots v_{i_m}$

> The previous example does not have a solution when viewed as an MPCP problem.

> So MPCP is indeed a different problem to PCP, but...

**Theorem 7.7.1**

*MPCP reduces to PCP*
Outline of Proof of Theorem 7.7.1

> Given lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$ for MPCP, suppose that symbols $\Diamond, \triangle$ are not in the strings.

> Construct lists $C = (w_1, \ldots, w_{k+2})$ and $D = (x_1, \ldots, x_{k+2})$ for PCP as follows.
  > For $i = 1, \ldots, k$, if $u_k = s_1 \ldots s_\ell$, then $w_{k+1} = s_1 \Diamond s_2 \Diamond \cdots \Diamond s_\ell \Diamond$. [$\Diamond$ succeeds symbols]
  > For $i = 1, \ldots, k$, if $v_k = s_1 \ldots s_\ell$, then $x_{k+1} = \Diamond s_1 \Diamond s_2 \Diamond \cdots \Diamond s_\ell$. [$\Diamond$ precedes symbols]
  > $w_1 = \Diamond w_2$ and $x_1 = x_2$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  > $w_{k+2} = \triangle$ and $x_{k+2} = \triangle$. [Balances the extra $\Diamond$]

\[
\begin{array}{cccc}
A & C & B & D \\
\begin{array}{c}
110 \\
0011 \\
0110 \\
\end{array} & \begin{array}{c}
\Diamond 1 \Diamond 0 \\
1 \Diamond 0 \\
0 \Diamond 1 \Diamond 1 \\
0 \Diamond 1 \Diamond 0 \\
\end{array} & \begin{array}{c}
110110 \\
00 \\
110 \\
\end{array} & \begin{array}{c}
\Diamond 1 \Diamond 0 \Diamond 1 \Diamond 0 \\
\Diamond 0 \\
\Diamond 1 \Diamond 0 \\
\Diamond \triangle \\
\end{array}
\end{array}
\]

$w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$ (PCP)

$u_1 u_{i_2-1} \cdots u_{i_m-1-1} = v_1 v_{i_2-1} \cdots v_{i_m-1-1}$ (MPCP)
**PCP is undecidable**

**Theorem 7.7.2**

*PCP is undecidable.*

**Outline of Proof of Theorem 7.7.2**

- The proof proceeds by constructing a MPCP for each TM $M$ and input $w$
- WLOG, we may assume the TM is 1-sided.

**Rule A:** Construct two lists $A$ and $B$ whose first entries are $\diamond$ and $\diamond q_0 w \diamond$

**Rule B:** Suppose $q$ is not a final state. Then, append to the list the following entries

<table>
<thead>
<tr>
<th>List $A$</th>
<th>List $B$</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$qX$</td>
<td>$Yp$</td>
<td>if $\delta(q, X) = (p, Y, R)$</td>
</tr>
<tr>
<td>$ZqX$</td>
<td>$pZY$</td>
<td>if $\delta(q, X) = (p, Y, L)$</td>
</tr>
<tr>
<td>$q \diamond$</td>
<td>$Yp \diamond$</td>
<td>if $\delta(q, B) = (p, Y, R)$</td>
</tr>
<tr>
<td>$Zq \diamond$</td>
<td>$pZY \diamond$</td>
<td>if $\delta(q, B) = (p, Y, L)$</td>
</tr>
</tbody>
</table>

**Rule C:** For $q \in F$, let $(XqY, q)$, $(Xq, q)$ and $(qY, Y)$ be corresponding pairs for $X, Y \in \Gamma$

**Rule D:** For $q \in F$ $(q \diamond \diamond, \diamond)$ is a corresponding pair.
PCP is undecidable

Outline of Proof of Theorem 7.7.2

- Suppose there is a solution to the MPCP problem. The solution starts with the first corresponding pair, and the string constructed from List $B$ is already a ID of TM $M$ ahead of the string from List $A$.
- As we select strings from List $A$ (corresponding to Rule B) to match the last ID, the string from List $B$ adds to its string another valid ID.
- The sequence of IDs constructed are valid sequences of IDs for $M$ starting from $q_0w$.
- Suppose the last ID constructed in the string constructed from List $B$ corresponds to a final state, then we can gobble up one neighboring symbol at a time using Rule C.
- Once we are done gobbling up all tape symbols, the string from List $B$ is still one final state symbol ahead of List $A$’s string.
- We then use Rule D to match and complete.

\[
\begin{align*}
\diamond & \text{String from List } B \text{ one ID ahead} \\
\diamond q_0w & \text{ Rule A} \\
\diamond q_0w & \text{ ‘} ID_1 \text{’} \\
\diamond ‘ID_1’ & \text{ Rule B} \\
\diamond ‘ID_2’ & \text{ } \ldots \text{ } ‘ID_k’ \\
S_1 & q_f S_4 S_5 \\
q_f S_5 & \text{ Rule C} \\
q_f & \text{ Rule D}
\end{align*}
\]

\[ID_k = s_1 s_2 q_f s_3 s_4 s_5\]
PCP is undecidable

Outline of Proof of Theorem 7.7.2

1. $M$ accepts $w \iff$ a solution to the MPCP exists.
2. If MPCP were decidable, then $L_u$ would be recursive, which it isn’t.
3. Hence, MPCP is undecidable. [Theorem 7.5.1]
4. Since MPCP is undecidable, PCP is also undecidable. [Theorem 7.5.1]
Ambiguity in CFGs
We’ll now revisit CFGs and prove that ambiguity in CFGs is undecidable.

**Theorem 7.8.1**

*The problem if a grammar is ambiguous is undecidable*

**Outline of Proof of Theorem 7.7.2**

- We’ll reduce every instance of PCP problem to a CFG.
- Given an PCP problem $A = (w_1, \ldots, w_k)$ and $B = (x_1, \ldots, x_k)$, pick symbols $a_1, \ldots, a_k$ that don’t appear in any string in list $A$ or $B$.
- Now define a grammar $G$ with production rules:

  $S \rightarrow A | B$
  $A \rightarrow w_1 A a_1 | \ldots | w_k A a_k | w_1 a_1 | \ldots | w_k a_k$
  $B \rightarrow x_1 B a_1 | \ldots | x_k B a_k | x_1 a_1 | \ldots | x_k a_k$

- If there are two leftmost derivations of a string in $L(G)$, one must use $S \rightarrow A$ and other $S \rightarrow B$.
- Every solution to the PCP leads to 2 leftmost derivations of some string in $L(G)$ and vice versa.
- Since PCP is undecidable, the ambiguity of CFGs must be undecidable [Thm 7.5.1]
Some More Undecidable Problems Concerning CFGs

> Given CFGs $G_1$ and $G_2$, is $L(G_1) \cap L(G_2) = \emptyset$?

> Given CFGs $G_1$ and $G_2$, is $L(G_1) \subseteq L(G_2)$?

> Given CFGs $G_1$ and $G_2$, is $L(G_1) = L(G_2)$?

> Given CFG $G$ and regular language $L$, is $L(G) = L$?

> Given CFG $G$ and regular language $L$, is $L \subseteq L(G)$?

> Given CFG $G$, is $L(G) = \Sigma^*$?