Decidability (What, stuff is unsolvable?)

CSCI 2670

University of Georgia

Fall 2014
Outline

- Decidability
- Decidable Problems for Regular Languages
- Decidable Problems for Context Free Languages
- The Halting Problem
- Countable and Uncountable Sets
- Diagonalization
- Unrecognizable Problems
A **Turing machine** is a 7-tuple \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\) such that

1. \(Q\) is a finite set of states;
2. \(\Sigma\) is the input alphabet, and \(\emptyset \notin \Sigma\);
3. \(\Gamma\) is the tape alphabet, where \(\emptyset \in \Gamma\) and \(\Sigma \subseteq \Gamma\);
4. \(\delta : (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})\) is the transition function;
5. \(q_0 \in Q\) is the start state;
6. \(q_{\text{accept}} \in Q\) is the accept state;
7. \(q_{\text{reject}} \in Q\) is the reject state, and \(q_{\text{reject}} \neq q_{\text{accept}}\).
Configurations

**Definition**

The state, tape contents, and tape head position constitute a **configuration**. An **accepting configuration** (rejecting configuration) is one in which the machine is in state $q_{accept}$ ($q_{reject}$). The **start configuration** has the tape head in the leftmost position, and the input string is written on it.

- A string $01010q_7010111$ may be used to indicate a configuration.
- The transition function maps one configuration to another.

**Definition**

A Turing Machine $M$ accepts a string $w$ if there exists a sequence of configurations $C_1, C_2, \ldots C_n$ such that

1. $C_1$ is the starting configuration of $M$ on input $w$,
2. $C_n$ is an accepting configuration,
3. for each $1 \leq i < n$, $C_i$ yields $C_{i+1}$.

$L(M)$, **language of** $M$ is the set of strings accepted by $M$. 
Definition

A language $L$ is **Turing–recognizable (recursively enumerable)** if there is a Turing machine $M$ such that $L(M) = L$.

- It is possible for a TM to never reach a halting configuration. On any given input $w$, it might instead loop.
  - These are the three options for any input: accept, reject, loop.
- A TM $M$ that halts on every input is called a **decider**.
- If $L(M) = L$, then $M$ is said to **decide** $L$.

Definition

A language $L$ is **Turing–decidable** (also called recursive) if there is a Turing machine $M$ that decides $L$. 
The definitions indicate that not all languages are decidable.

The Church-Turing Thesis equates “algorithmically solvable” with solvable by Turing machines.

- A yes/no problem is algorithmically decidable if there is some Turing machine that will decide it.
- Given any instance of the problem (input), the machine will always halt in finite time with the correct answer (yes or no).
- A problem is not algorithmically solvable if every TM loops on at least one input.

Chapter 4 discusses problems that are solvable/unsolvable by TMs.

Knowing which problems are unsolvable is important (and pretty cool).

- Why try to write a program to solve a problem when someone’s already told you it can’t be done.
Decidability and Undecidability

“I can’t find an ... algorithm, I guess I’m just too lame.”

► image: http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html
Decidability and Undecidability

“I can’t find an ... algorithm, because no such algorithm is possible.”

- **image**: [http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html](http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html)
Decidable Problems for Regular Languages

The following languages/problems are all decidable.

- \( A_{DFA} = \{ \langle B, w \rangle | B \text{ is a DFA and } w \in L(B) \} \)
- For any DFA \( B \), \( L(B) \) is decidable.
- \( A_{NFA} = \{ \langle B, w \rangle | B \text{ is an NFA and } w \in L(B) \} \)
- \( A_{REX} = \{ \langle R, w \rangle | R \text{ is a regular expression that generates } w \} \)
- \( E_{DFA} = \{ \langle B \rangle | B \text{ is a DFA and } L(B) = \emptyset \} \)
- \( EQ_{DFA} = \{ \langle A, B \rangle | A, B \text{ are DFAs and } L(A) = L(B) \} \)

- In each case, the input is a string representing, for instance, a DFA and an input string \( w \).
- We design a TM to decide the problem (typically by simulating the machines described in the input).
- A reasonable representation of the DFA is needed. E.g.,

\[
q_0, \ldots, q_n \# a_1, \ldots, a_m \# (q_0, a_0, q_i), \ldots \# q_0 \# q_i, q_j, \ldots \# w_0 \ldots w_k
\]
Theorem

\(A_{\text{DFA}}\) is decidable, where \(A_{\text{DFA}} = \{ \langle B, w \rangle | B \text{ is a DFA and } w \in L(B) \}\)
Theorem

\[ A_{DFA} \text{ is decidable, where } A_{DFA} = \{ \langle B, w \rangle | B \text{ is a DFA and } w \in L(B) \} \]

(Proof idea)

- We construct a Turing machine \( M \) to decide the problem.
- Scan the input string \( \langle B, w \rangle \), determining whether the input constitutes a valid DFA and string \( w \). If not, then reject.
- Scan the input string repeatedly.
  - Keep track of the current DFA state and position of \( w \).
  - Find the appropriate transition to update the state appropriately.
- After processing \( w_k \), check whether \( B \) is in an accept state.
  - If so, then accept \( \langle B, w \rangle \). Otherwise reject \( \langle B, w \rangle \).
Theorem

If $B$ is a DFA, then $L(B)$ is decidable.

- This is clearly true, as every DFA halts on every input.
- We can easily create a TM $M_B$ to recognize $L(B)$ by modifying the transition function of $B$ to fit the form for TMs.
  - If $\delta_B(q, a) = q'$, then $\delta_{M_B}(q, a) = (q', a, R)$.

Alternatively, we can use $M$ from the previous proof in $M_B$.

- On input $w$, run $M$ on $\langle B, w \rangle$.
- If $M$ accepts $\langle B, w \rangle$, accept $w$.
- Otherwise reject $w$. 
Theorem

$A_{NFA} \text{ is decidable, where } A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA and } w \in L(B) \}$
Theorem

\( A_{NFA} \) is decidable, where \( A_{NFA} = \{ \langle B, w \rangle | B \text{ is an } NFA \text{ and } w \in L(B) \} \)

(Proof idea)

- We construct another Turing machine \( N \) to decide \( A_{NFA} \):
  - Scan the input string \( \langle B, w \rangle \), determining whether the input constitutes a valid NFA and string \( w \). If not, then reject.
  - Convert \( B \) to a DFA \( C \) (using the construction described in Ch 1).
  - Run \( M \) on \( \langle C, w \rangle \).
  - Accept \( \langle B, w \rangle \) if \( M \) accepts \( \langle C, w \rangle \). Otherwise reject \( \langle B, w \rangle \).
Decidable Problems for Regular Languages

Theorem

\( A_{REX} \) is decidable, where

\[
A_{REX} = \{ \langle R, w \rangle | R \text{ is a regular expression that generates } w \}\]
Decidable Problems for Regular Languages

Theorem

\[ A_{REX} \text{ is decidable, where} \]

\[ A_{REX} = \{ \langle R, w \rangle | R \text{ is a regular expression that generates } w \} \]

(Proof idea)

- We construct another Turing machine \( P \) to decide \( A_{REX} \):
- Scan the input string \( \langle R, w \rangle \), determining whether the input constitutes a valid reg-exp and string \( w \). If not, then reject.
- Convert \( R \) to an NFA \( A \) (using the construction described in Ch 1).
- Run \( N \) on \( \langle A, w \rangle \).
- Accept \( \langle R, w \rangle \) if \( N \) accepts \( \langle A, w \rangle \). Otherwise reject \( \langle R, w \rangle \).
Decidable Problems for Regular Languages

**Theorem**

$E_{DFA}$ is decidable, where

$$E_{DFA} = \{ \langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset \}$$

Idea: Check whether an accept state is reachable from $A$’s start state.
Theorem

\[ E_{DFA} \text{ is decidable, where} \]

\[ E_{DFA} = \{ \langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset \} \]

Idea: Check whether an accept state is reachable from \( A \)'s start state.

(Proof idea)

- We construct Turing machine \( T \) to decide \( E_{DFA} \):
- Scan the input string \( \langle A \rangle \), determining whether the input constitutes a valid DFA. If not, then reject.
- We assume states of \( A \) are listed left to right, with \( q_0 \) being leftmost.
- Mark \( q_0 \). Using the transition function of \( A \), mark any unmarked state \( q' \) such that \( \delta(q, a) = q' \) and \( q \) is already marked.
- Continue until no new nodes can be marked.
- If an accept state is marked, then reject \( \langle A \rangle \). Otherwise accept it.
Decidable Problems for Regular Languages

Theorem

$EQ_{DFA}$ is decidable, where $EQ_{DFA} = \{ \langle A, B \rangle | A, B $ are DFAs and $L(A) = L(B) \}$

- Regular languages are closed under complementation, union, intersection.
- Given $A$ and $B$, we can construct a DFA for the symmetric difference:
  \[
  Diff = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)})
  \]
- If $Diff$ is nonempty, then $L(A) \neq L(B)$. 
Decidable Problems for Regular Languages

**Theorem**

$EQ_{DFA}$ is decidable, where $EQ_{DFA} = \{ \langle A, B \rangle | A, B \text{ are DFAs and } L(A) = L(B) \}$

- Regular languages are closed under complementation, union, intersection.
- Given $A$ and $B$, we can construct a DFA for the symmetric difference:
  \[
  Diff = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)})
  \]
- If $Diff$ is nonempty, then $L(A) \neq L(B)$.

**Proof idea**

- We construct a Turing machine $F$ to decide $EQ_{DFA}$:
  - Scan the input string $\langle A, B \rangle$ determining whether the input constitutes two valid DFAs $A$ and $B$. If not, then reject.
  - Construct a string representing the DFA $C$, where $L(C)$ is the symmetric difference of $L(A)$ and $L(B)$.
  - Run Turing Machine $T$ on $\langle C \rangle$. (Recall that $T$ decides $E_{DFA}$.)
  - If $T$ accepts $\langle C \rangle$, accept $\langle A, B \rangle$. Otherwise reject it.
Decidable Problems for Regular Languages

Answer all questions for the following DFA $M$ and give reasons for your answers.

1. Is $\langle M, 0110 \rangle \in A_{DFA}$?
2. Is $\langle M, 001 \rangle \in A_{DFA}$?
3. Is $\langle M \rangle \in A_{DFA}$?
4. Is $\langle M, 0110 \rangle \in A_{REX}$?
5. Is $\langle M \rangle \in E_{DFA}$?
6. Is $\langle M, M \rangle \in EQ_{DFA}$?
Decidable Problems for Context Free Languages

The following languages/problems are decidable.

- \( A_{CFG} = \{ \langle G, w \rangle | G \text{ is a CFG generating } w \} \)
- \( E_{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(A) = \emptyset \} \)
- If \( G \) is a CFG, then \( L(G) \) is decidable.

The following language/problem is **NOT** decidable.

- \( EQ_{CFG} = \{ \langle G, H \rangle | G, H \text{ are CFGs and } L(G) = L(H) \} \)

Why won't using \((L(G) \cap \overline{L(H)}) \cup (L(H) \cap \overline{L(G)})\) work in this case?
The following language/problem is **NOT** decidable.

\[ EQ_{CFG} = \{ \langle G, H \rangle | G, H \text{ are CFGs and } L(G) = L(H) \} \]

Context free grammars are not closed under complementation or intersection, and so we cannot use

\[(L(G) \cap \overline{L(H)}) \cup (L(H) \cap \overline{L(G)})\]

as was done for \( EQ_{DFA} \).
Theorem

$A_{\text{CFG}}$ is decidable, where $A_{\text{CFG}} = \{ \langle G, w \rangle | G \text{ is a CFG generating } w \}$. 
Theorem

$A_{CFG}$ is decidable, where $A_{CFG} = \{ \langle G, w \rangle | G \text{ is a CFG generating } w \}$. 

Idea: If $G$ is in CNF, then it takes at most $2n - 1$ steps to generate $w$.

(Proof idea)

- We construct a Turing machine $S$ to decide the problem.
- Scan the input string $\langle G, w \rangle$, determining whether the input constitutes a valid CFG and string $w$. If not, then reject.
- Convert $G$ to CNF, using the procedure described in Ch 2.
- If $w = \varepsilon$, look for grammar rule $S \rightarrow \varepsilon$. If present, then accept. Otherwise reject.
- If $w \neq \varepsilon$, systematically generate all derivations of at most $2n - 1$ steps, where $|w| = n$.
- If any of these derivations produce $w$, then accept. If not, reject.
Theorem

\( E_{CFG} \text{ is decidable, where } E_{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset \} \).
Theorem

$E_{CFG}$ is decidable, where $E_{CFG} = \{ \langle G \rangle | G$ is a CFG and $L(G) = \emptyset \}$.

Idea: Work backwards from terminal symbols to start symbol $S$.

(Proof idea)

- We construct a Turing machine $R$ to decide the problem.
- Scan the input string $\langle G \rangle$, determining whether the input constitutes a valid CFG. If not, then reject.
- Convert $G$ to Chomsky Normal Form, as described in Ch 2.
- For each rule $A \to a$, mark every occurrence of $A$ in the rules.
- For each rule of the form $A \to BC$, if both $B$ and $C$ are marked, then mark every occurrence of $A$ in the rules.
- Continue until no new variables are marked.
- If $S$ is NOT marked, then accept $\langle G \rangle$. Otherwise Reject.
Decidable Problems for Context Free Languages

Theorem

$E_{CFG}$ is decidable, where $E_{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset \}$.

An easier alternative.

(Proof idea)

- We construct a Turing machine $R$ to decide the problem.
- Scan the input string $\langle G \rangle$, determining whether the input constitutes a valid CFG. If not, then reject.
- Mark every terminal in every rule.
- For each rule of the form $A \rightarrow w$, if every symbol of $w$ is marked, mark every occurrence of $A$ in the rules.
- Continue until no new variables are marked.
- If $S$ is NOT marked, then accept $\langle G \rangle$. Otherwise Reject.
Theorem

*If G is a CFG, then \( L(G) \) is decidable.*
Decidable Problems for Context Free Languages

Theorem

If $G$ is a CFG, then $L(G)$ is decidable.

We can use the machine $S$, which decides $A_{CFG} = \{ \langle H, w \rangle | H$ is a CFG generating $w \}$, to construct a TM $M_G$ to decide $L(G)$.

- On input $w$, run $S$ on $\langle G, w \rangle$.
- If $S$ accepts $\langle G, w \rangle$, accept $w$.
- Otherwise reject $w$. 
Closure Property

<table>
<thead>
<tr>
<th>Operations</th>
<th>RL</th>
<th>CFL</th>
<th>T-Recog</th>
<th>T-Deci</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Concatenation</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Star</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Intersection</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Complementation</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

- **RL**: Regular Languages; **CFL**: Context-Free Languages; **T-Recog**: Turing-Recognizable Languages; **T-Deci**: Turing Decidable Languages.
- The intersection of a CFL and a RL is a CFL.
- It is recommended by Ian Lindsey Berrigan.
There are problems/languages that are undecidable.

In general, program verification (deciding whether a program runs "correctly") is undecidable.

The Halting Problem: The most famous undecidable problem in CS.

Theorem

\[ A_{TM} \text{ is undecidable, where} \]

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} . \]

The proof of this is a proof by contradiction.
Before we get to the proof, let us first observe that $A_{TM}$ is Turing-recognizable.

Thus this theorem shows that recognizers are more powerful than deciders.

The following TM $U$ recognizes $A_{TM}$.

$U =$ “On input $\langle M, w \rangle$, where $M$ is a TM and $w$ is a string:

1. Simulate $M$ on input $w$.

2. If $M$ ever enters its accept state, accept; if $M$ ever enters its reject state, reject.”

Note that this machine loops on $\langle M, w \rangle$ if $M$ loops on $w$. Hence $A_{TM}$ sometimes called the halting problem.
The Halting Problem

**Theorem**

\[ A_{TM} \text{ is undecidable, where} \]

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} . \]

**Proof.**

Suppose \( A_{TM} \) is decidable. Then there’s a machine \( H \) deciding it.

- The input of \( H \) is a string \( \langle M, w \rangle \).
- \( M \) is a TM, and \( w \) is a string.
- \( H \) runs \( M \) on \( w \).
  - If \( M \) accepts \( w \), then \( H \) stops and accepts.
  - If \( M \) does not accept \( w \), then \( H \) stops and rejects.

Observe that there is no string on which \( H \) does not halt.
The Halting Problem

continued.

Given \( H \), we construct a new Turing machine \( D \):

- The input of \( D \) is a string \( \langle M \rangle \).
- \( D \) runs \( H \) on \( \langle M, \langle M \rangle \rangle \).
  - If \( H \) accepts \( \langle M, \langle M \rangle \rangle \), then \( D \) rejects \( \langle M \rangle \).
  - If \( H \) rejects \( \langle M, \langle M \rangle \rangle \), then \( D \) accepts \( \langle M \rangle \).

Given the definition of \( H \) and \( D \):

- On input \( \langle M \rangle \),
  - \( D \) accepts \( \langle M \rangle \) if \( M \) does not accept \( \langle M \rangle \).
  - \( D \) rejects \( \langle M \rangle \) if \( M \) accepts \( \langle M \rangle \).
The Halting Problem

Given the definition of $H$ and $D$:

- On input $\langle M \rangle$,
  - $D$ accepts $\langle M \rangle$ if $M$ does not accept $\langle M \rangle$.
  - $D$ rejects $\langle M \rangle$ if $M$ accepts $\langle M \rangle$.

On input $\langle D \rangle$, we then find

- $D$ accepts $\langle D \rangle$ if $D$ does not accept $\langle D \rangle$.
- $D$ rejects $\langle D \rangle$ if $D$ accepts $\langle D \rangle$.

Under the assumption that $A_{TM}$ is decidable, there is a machine $D$ which simultaneously accepts and rejects its own specification $\langle D \rangle$.

This is a contradiction!

And so the assumption that $A_{TM}$ is decidable is false.

We conclude that $A_{TM}$ is undecidable.

□
Countability and Uncountability/Diagonalization

- Implicitly, the proof that \( A_{TM} \) is undecidable relies on Cantor’s diagonalization method.
- This is used to prove that a set is uncountable.

- Recall that a set \( S \) is countable if either it is finite or there is a one-to-one correspondence between elements of \( S \) and those of \( \mathbb{N} \).
- That is, there is a bijective function \( f \) from \( \mathbb{N} \) to \( S \).
  - injective: For all \( a, b \in \mathbb{N} \), if \( a \neq b \), then \( f(a) \neq f(b) \).
  - surjective: For all \( b \in S \), there is an \( a \in \mathbb{N} \) such that \( f(a) = b \).
- Intuitively, \( S \) is countable if we can write the elements of \( S \) as a list, and each element of \( S \) appears exactly once.

- A set \( A \) is countable if either it is finite or it has the same size as \( \mathbb{N} \).
- The set of positive even integers is countable, as is the set of positive odd numbers.
- The set of positive rational numbers is countable as well.
The set of finite strings $S$ over any finite alphabet $\Sigma$ is countably infinite.

- Assume an ordering of symbols in $A$.
- The strings of $S$ can be ordered as follows:
  - For any $n \geq 0$, all strings of length $n$ come before all those of length $n + 1$.
  - For any given $n$, arrange the strings of length $n$ in according to the ordering of the alphabet symbols (as in a dictionary).
- Crucially, for any $n$, only a finite number of strings have length $n$.

- As a corollary to this, the set of Turing machines is countably infinite.
- Each TM can be written as a finite string over a given alphabet.
In contrast, the set of infinite strings $S$ over a finite alphabet $\Sigma$ is not countable.

We’ll prove this using Cantor’s diagonalization method, a type of proof by contradiction. Assume $\Sigma = \{0, 1\}$ as the alphabet.

An infinite binary sequence is an unending sequence of 0s and 1s.

Each element of $S$ is an infinitely long string of 0s and 1s.

- E.g., 01010101111010010111000101 ...
- We will refer to particular bits of a string: $b_1 b_2 b_3 \ldots$

Theorem

*The Set $B$ of infinitely long strings over $\Sigma = \{0, 1\}$ is uncountable.*
The set of infinite strings are uncountable

Theorem

The Set $B$ of infinite strings over $\Sigma = \{0, 1\}$ is uncountable.

Proof.

- Assume $B$ is countable.
- As such, the elements of $B$ can be listed as a sequence $s_1, s_2, \ldots$ (there's a first element, a second element, a third ...).
- Every element of $B$ appears exactly once in the sequence.
  - $s_1 = b_{1,1} b_{1,2} b_{1,3} b_{1,4} b_{1,5} \ldots$
  - $s_2 = b_{2,1} b_{2,2} b_{2,3} b_{2,4} b_{2,5} \ldots$
  - $s_3 = b_{3,1} b_{3,2} b_{3,3} b_{3,4} b_{3,5} \ldots$
  - $\vdots$
- Above, $b_{i,j}$ refers to the $j$th bit of the $i$th element of $B$. 
The set of infinite strings are uncountable

**Theorem**

*The Set $B$ of infinite strings over $\Sigma = \{0, 1\}$ is uncountable.*

**Proof, continued.**

- We construct an infinite bitstring $X = c_1 c_2 c_3 \ldots$ by examining the diagonal:
  - $s_1 = b_{1,1} b_{1,2} b_{1,3} b_{1,4} b_{1,5} \ldots$
  - $s_2 = b_{2,1} b_{2,2} b_{2,3} b_{2,4} b_{2,5} \ldots$
  - $s_3 = b_{3,1} b_{3,2} b_{3,3} b_{3,4} b_{3,5} \ldots$
  - $\vdots$
  - For each $i \geq 1$, if $b_{i,i} = 1$, then $c_i = 0$. Otherwise $c_i = 1$.
  - For instance:
    - If $b_{1,1} b_{2,2} b_{3,3} b_{4,4} b_{5,5} \ldots = 01101\ldots$
    - Then $c_1 c_2 c_3 c_4 c_5 \ldots = 10010\ldots$
  - Observe that for any $k$, $b_{k,k} \neq c_k$. 
Theorem

The Set $B$ of infinite strings over $\Sigma = \{0, 1\}$ is uncountable.

Proof, continued.

- We construct an infinite bitstring $X = c_1c_2c_3\ldots$:
  - If $b_{1,1}b_{2,2}b_{3,3}b_{4,4}b_{5,5}\ldots = 01101\ldots$
  - Then $c_1c_2c_3c_4c_5\ldots = 10010\ldots$

- Observe that $X$ is an infinite bitstring and so must be some $s_n$ on the list (it must appear somewhere on the list).

- And so $b_{n,n}$ (the nth bit of $s_n$) is the same as $c_n$.

- However, by construction of $X$, $b_{n,n} \neq c_n$.

- A contradiction!

- And so the assumption that $B$ is countable must be false.

$\square$
For any finite $\Sigma$, $\Sigma^*$ is a set of finite strings and so is countable.

That is, we can list out the strings of $\Sigma^*$: $s_1, s_2, s_3, \ldots$.

Any language $L$ over $\Sigma$ can thus be represented as an infinite bitstring $b_1b_2b_3\ldots$.

- $s_i \in L$ if and only if $b_i = 1$.
- $s_i \notin L$ if and only if $b_i = 0$.

The string $b_1b_2b_3\ldots$ is called the characteristic sequence for $L$.

It is clear that, for any $\Sigma$, there is a one-to-one correspondence between the languages over $\Sigma$ and the set of infinite bit-strings.

Given the previous proof, the following may be inferred.

**Theorem**

*For any $\Sigma$, the set of languages over $\Sigma$ is uncountable.*

**Corollary**

*There exist languages that are not Turing Recognizable.*
The set of Turing Machines is countable, and so we can list them.

The set of finite strings over $\Sigma$ is also countable.

We then can construct the following table.

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$M_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$M_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$M_4$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

A 1 in a cell $i,j$ means that $M_i$ accepts $\langle M_j \rangle$.

A 0 in a cell $i,j$ means that $M_i$ does not accept $\langle M_j \rangle$.

And so the table gives a partial description of $H$. 

Recall $D$: On input $\langle M_i \rangle$,

- $D$ accepts $\langle M_i \rangle$ if $M_i$ does not accept $\langle M_i \rangle$.
- $D$ rejects $\langle M_i \rangle$ if $M_i$ accepts $\langle M_i \rangle$.

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

So, the output of $D$ can be found by flipping bits in the diagonal.
Recall $D$: On input $\langle M_i \rangle$,

- $D$ accepts $\langle M_i \rangle$ if $M_i$ does not accept $\langle M_i \rangle$.
- $D$ rejects $\langle M_i \rangle$ if $M_i$ accepts $\langle M_i \rangle$.

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>...</th>
<th>$\langle D \rangle$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>$D$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>???</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

So, the output of $D$ can be found by flipping bits in the diagonal.

But $D$ must be equal to some $M_n$ and so appear on the list.

At cell $(n, n)$, the contents is both 1 and 0. A contradiction!
**Definition**

Language $L$ is **co-Turing recognizable (co-recursively enumerable)** if it is the complement of a Turing recognizable language.

**Theorem**

$L$ is Turing decidable if and only if $L$ is recognizable and co-recognizable.

**Proof.**

- If $L$ is decidable, then it is clearly recognizable. Also, if $M$ is the machine deciding $L$, we construct a new machine $M'$ to decide $\overline{L}$ by simply doing the opposite of $M$. So $M'$ recognizes $\overline{L}$. As such, $L$ is co-recognizable.

- Now assume that $L$ is recognizable and co-recognizable. Then $\overline{L}$ is recognizable, and TMs $M_1$ and $M_2$ recognize $L$ and $\overline{L}$. We construct TM $M$ deciding $L$. On input $w$, $M$ runs $M_1$ and $M_2$ in parallel. If $M_1$ accepts $w$, then $M$ accepts. If $M_2$ accepts $w$, then $M$ rejects. It is clear that it accepts all strings of $L$ and reject all those not in $L$. 
Unrecognizable Languages

Theorem

L is Turing decidable if and only if L is recognizable and co-recognizable.

- \( A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} \) is undecidable.
- \( A_{TM} \) is clearly recognizable, however. \( H \) runs \( M \) on \( w \) and accepts if and only if \( M \) accepts \( w \).
- From this, the below holds.

Theorem

\( \overline{A_{TM}} \) is not recognizable.

Proof.

If \( \overline{A_{TM}} \) is recognizable, then a machine \( M \) recognizes it. We can construct a machine \( N \) to decide \( A_{TM} \) by running \( M \) and \( H \) simultaneously. On any input \( w \), one of them must accept \( w \). If \( H \) accepts \( w \), then \( N \) accepts. If \( M \) accepts, then \( N \) rejects. But \( A_{TM} \) is not decidable, and so no such machine \( M \) exists.
Show the following languages/problems are decidable.

- \( EQ_{DFA,REX} = \{ \langle A, R \rangle | A \text{ is a DFA, } R \text{ is a regular expression and } L(A) = L(R) \} \)
- \( A = \{ \langle M \rangle | M \text{ is a DFA which does not accept any string containing an odd number of 1s } \} \)
- \( A_{\varepsilon-CFG} = \{ \langle G \rangle | G \text{ is a CFG that generates } \varepsilon \} \)
More Decidable Problems

Show the following languages/problems are decidable.

- $S = \{ \langle M \rangle | M \text{ is a DFA that accepts } w^R \text{ whenever it accepts } w \}$
- $ALL_{DFA} = \{ \langle A \rangle | A \text{ is a DFA and } L(A) = \Sigma^* \}$
- $A = \{ \langle R, S \rangle | R \text{ and } S \text{ are regular expressions and } L(R) \subseteq L(S) \}$
- $A = \{ \langle R \rangle | R \text{ is a regular expression describing a language containing at least one string } w \text{ that has } 111 \text{ as a substring (i.e., } w = x111y \text{ for some } x \text{ and } y \}$
Show the following languages/problems are decidable.

- $INFINITE_{DFA} = \{ \langle A \rangle | A \text{ is a DFA and } L(A) \text{ is an infinite language} \}$
- $INFINITE_{PDA} = \{ \langle B \rangle | B \text{ is a PDA and } L(B) \text{ is an infinite language} \}$
INFINITE\text{\textsubscript{DFA}} is decidable, where

\[ \text{INFINITE}\text{\textsubscript{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) \text{ is an infinite language}\} \].

Proof.

The following TM \( X \) decides \text{INFINITE}\text{\textsubscript{DFA}}.

\( X = \) “On input \( \langle A \rangle \) where \( A \) is a DFA:

1. Let \( k \) be the number of states of \( A \).
2. Construct a DFA \( D \) that accepts all strings of length \( k \) or more.
3. Construct a DFA \( M \) such that \( L(M) = L(A) \cap L(D) \).
4. Test \( L(M) = \emptyset \), using the \( E_{DFA} \) decider \( T \).
5. If \( T \) accepts, reject; if \( T \) rejects, accept.”
Theorem

\[ \text{INFINITE}_{PDA} \text{ is decidable, where} \]

\[ \text{INFINITE}_{PDA} = \{ \langle B \rangle | B \text{ is a PDA and } L(B) \text{ is an infinite language} \}. \]

Proof.

The following TM \( Y \) decides \( \text{INFINITE}_{PDA} \).

\( Y = \) “On input \( \langle B \rangle \) where \( B \) is a PDA:

1. Convert \( B \) to a CFG \( G \) and compute \( G \)'s pumping length \( p \).
2. Construct a regular expression \( E \) that contains all strings of length \( p \) or more.
3. Construct a CFG \( H \) such that \( L(H) = L(G) \cap L(E) \).
4. Test \( L(H) = \emptyset \), using the \( E_{CFG} \) decider \( R \).
5. If \( R \) accepts, reject; if \( R \) rejects, accept.”