Chapter 5
Decidability

We have introduced Turing Machine as a “standard” model for the general computation and defined the informal notion of algorithms in terms of Turing Machine. Now, we want to investigate the power of such a model in terms of its computability, namely, what can be done, and what cannot be done with this model.

For example, we will show that there exists an algorithm that will decide if a CFL will generate a specific string, which is the centerpiece of a compiler, i.e., the acceptance problem dealing with the context-free language.

Since every regular language is context free, we immediately know that the acceptance problem dealing with regular languages is solvable.
The other side

We will also show that there doesn’t exist any algorithm, or equivalently, a Turing machine, that will tell us if an arbitrary TM will accept an arbitrary string. That is, the acceptance problem dealing with a universal language, or TM, is unsolvable.

This opens up the can....

Just like what we did with the NP-complete problem, once a problem is shown algorithmically unsolvable, we have to consider its simplification, which often leads us to a better understanding of its nature.
Decidable problems w.r.t. RL

The accepting problem for DFAs of testing if a particular DFA accepts a given string can be expressed as the following language: $A_{\text{DFA}} = \{ \langle B, w \rangle | B \text{ is a DFA that accepts } w. \}$. The notions of $A_{\text{NFA}}$ and $A_{\text{REX}}$ are similarly defined.

**Theorem:** $A_{\text{DFA}}$ is a decidable language.

**Proof:** For any input $\langle B, w \rangle$, where $B$ is a DFA and $w$ is a string:
1. Simulate $B$ on $w$.
2. If the simulation ends in an accept state, accept it; otherwise, rejects it.

Because we can convert any NFA, or a regular expression, to a DFA, thus, we have the following result:

**Theorem:** Both $A_{\text{NFA}}$, and $A_{\text{REX}}$ are decidable languages.
Let $E_{\text{DFA}}$ be $\{\langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset \}$. 

**Theorem:** $E_{\text{DFA}}$ is decidable.

**Proof:** A DFA accepts a string if and only if it is possible to go from the start state to an accept state. Thus, we can construct the following TM:

For any input $\langle A \rangle$, where $A$ is a DFA,

1. Mark the start state of $A$.

2. Repeat Step 3 until no new state gets marked

3. Mark any state that has a transition coming into it from a marked state.

4. If no accept state gets marked, accept; otherwise, reject.
Let $EQ_{DFA}$ be $\{\langle A, B \rangle | A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$.

**Theorem:** $EQ_{DFA}$ is decidable.

**Proof:** We will construct another DFA $C$, such that $C$ only accepts those strings that are accepted by either $A$ or $B$, but not by both. Then, $L(C) = \emptyset \equiv L(A) = L(B)$. Obviously,

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}).$$

Because both $L(A)$ and $L(B)$ are RL’s, and RLs are closed under union, intersection, as well as complementation, so is $L(C)$.

Thus, to decide if $L(A) = L(B)$, we simply construct $C$ and run $E_{DFA}$ on $C$.

**Homework:** Exercise 4.1-4.3.
déjà vu

[2.26.] If \( G \) is a CNF grammar, then for any string \( w \in L(G), |w| = n \), it takes exactly \( 2n - 1 \) steps to derive \( w \).

**Proof by induction on** \( n \): By definition, we have that

\[
S \Rightarrow^* w.
\]

For the base case. If \( n = 1 \), then, by the definition of Chomsky normal form, the only rule we can use must be either \( S \rightarrow \epsilon \), or \( S \rightarrow a \), \( a \in \Sigma \). Thus, \( w \equiv \epsilon \), or \( w \equiv a \), for some \( a \in \Sigma \). In both cases, we have that \( S \Rightarrow w \), i.e., \( w \) can be derived in just one step. Hence, the base case is proved.
Assume the result is true for \( n \), i.e., if \(|w| = n\), and \( S \Rightarrow^* w \), then it takes exactly \( 2n - 1 \) steps to derive \( w \). Let \(|w| = n + 1\) and \( S \Rightarrow^* w \). Then the first step must be \( S \Rightarrow BC \), by the definition of CNF. Hence, we have that

\[
S \Rightarrow BC \Rightarrow^* w_1 \cdot w_2 \equiv w.
\]

Let \(|w_i| = n_i, i \in 1, 2\). Assume that, e.g., \( B \Rightarrow^* \epsilon \). By the definition of CNF, it must be the case that for some \( D \in V, D \Rightarrow \epsilon \in R \). The latter means, by definition, that \( D \equiv S \). Then, \( S \) would have occurred on the right hand side of a rule, which is forbidden by the CNF grammar. Therefore, \( n_1 > 0 \) and \( n_2 > 0 \), i.e., \( n_i < n, i = 1, 2 \).

By inductive assumption, we have that it takes exactly \( 2n_1 - 1 \) and \( 2n_2 - 1 \) steps to derive \( w_1 \) and \( w_2 \) respectively. Adding the step to derive \( BC \) from \( S \), it takes exactly \( \lceil 2(n_1 + n_2) - 2 \rceil + 1 = 2n - 1 \) steps to derive \( w \).
Decidable problems w.r.t. CFL

Let $A_{\text{CFG}} = \{⟨G, w⟩|G \text{ is a CFG that accepts } w.\}$. 

**Theorem:** $A_{\text{CFG}}$ is a decidable language.

**Proof:** The idea to try all derivations does not work, in general, since if $G$ doesn’t generate $w$, the algorithm might never halt.

On the other hand, we can always convert $G$ to a Chomsky Normal Form, which can be done effectively as we saw in the last chapter.

If the latter generates $w$, as we just saw, it must generate $w$ in $2|w| − 1$ steps.
An “inefficient” compiler

For any input \( \langle G, w \rangle \), 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 \( 2|w| - 1 \) steps.

3. If any of them accepts \( w \), accept it; otherwise, reject.

Although Step 2 is extremely time consuming, it generates and checks at most \( b^2|w|^{-1} \) strings, thus can be done within a finite amount of time.

The rest is just an engineering story, which must be told in a course on compiler.
Let $E_{\text{CFG}}$ be $\{\langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset \}$.

**Theorem:** $E_{\text{CFG}}$ is decidable.

**Proof:** We can’t use the previous result to test for emptiness, by running each and every word with the grammar, as there are infinite number of strings over any alphabet.

The following is the algorithm:

For any input $\langle G \rangle$, a CFG,

1. Mark all terminals in $G$.

2. Repeat Step 3 until no new variable gets marked.

3. Mark any variable $A$, where $G$ contains a rule $A \rightarrow U_1 \cdots U_k$, and all $U_i$’s have been marked.

4. If the start symbol is not marked, accept; otherwise, reject. □
Let $EQ_{CFG}$ be $\{\langle G, H \rangle | G$ and $H$ are CFGs and $L(G) = L(H)\}$.

We can’t use the same idea as we used to show that $EQ_{DFA}$ is decidable. (Why?) In fact, $EQ_{CFG}$ is not decidable.

**Theorem:** Every CFL is decidable.

This result means that for any CFL, $L$, and any string $w$, there is a TM that decides if $w \in L$.

**Proof:** Let $G$ be a CFG for $L$, and let $S$ be the TM we used to decide the acceptance problem for CFL, we construct a TM, $M_G$ : For input $\langle G, w \rangle$,

1. Run TM $S$ on $\langle G, w \rangle$.

2. If $S$ accepts $\langle G, w \rangle$, accept; otherwise, reject.
Some positive results

So far, we have coined four classes of languages: regular, context-free, enumerable and decidable. The following chart shows their relationship.

![Chart showing the relationship between different classes of languages]

**Question:** Why the “⊂” relationship looks like this?

**Question:** Why not “⊆”?  

**Homework:** Exercises 4.4, and 4.13.
The Halting problem

We prove the existence of a concrete, and easily understood, problem that is algorithmically unsolvable, i.e., not decidable.

By the halting problem, we mean the following: given a Turing Machine, $M$, and an input string, $s$, whether $M$ accepts $s$?

Let $A_{TM} = \{ \langle M, w \rangle | M \text{ accepts } w \}$.

**Theorem:** $A_{TM}$ is undecidable.

In other words, there does not exist a TM which will decide $A_{TM}$, i.e., it is able to tell whether $M$ accepts $w$ or not.
It is enumerable....

...since there is TM that can accept $A_{TM}$.

For example, we can construct the following TM, $U$, the universal Turing Machine, for input $\langle M, w \rangle$, where $M$ is a TM and $w$ is an input string:

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

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

Notice that this universal machine that Turing suggested in 1936 shows the feasibility of a stored program computer, which was eventually implemented in the 1940's.
Diagonalization

This classic technique was originally used to investigate infinite sets. Given two finite sets, it is easy to see which is larger. But, it is not so easy to answer the same question for two sets that contain infinite amount of elements.

Georg Cantor (1845-1918) observed that, if two finite sets are of the same size, then there exists a 1-1 correspondence between the elements of these two sets and proposed to use the same criterion for infinite sets.

Assume that we have a function \( f : A \mapsto B \). Then, \( a = b \Rightarrow f(a) = f(b) \).

By saying that \( f \) is one-to-one (injective), we mean that \( f \) never maps two different elements to the same element, i.e., \( a \neq b \Rightarrow f(a) \neq f(b) \); \( f \) is onto (surjective), if \( f \) hits every element in \( B \), i.e., \( \forall b \in B, \exists a, f(a) = b \).
|A| = |B|

**Question:** When |A| = |B|, i.e., A and B are of the same size?

**Answer:** If there is a one to one, and onto function $f: A \mapsto B$.

By the pigeon hole principle, if |A| > |B|, for some $b \in B$, there exist at least two elements from $A$ that will be mapped to $b$. This violates the assumption that $f$ is injective, i.e., one-to-one. Hence, |A| ≤ |B|.

Just assume |A| < |B|. Then, since $f$ is a function, $f$ maps any element in $A$ to at most one element in $B$. Hence, there must exist at least one $b \in B$, such that no element from $A$ is mapped to $b$, which violates the onto condition.
An example

Let \( \mathcal{N} \) be the set of natural numbers, \( \{1, 2, \ldots\} \) and let \( \mathcal{E} \) be the set of all the even natural numbers, \( \{2, 4, \ldots\} \).

**Question:** Which set contains more numbers?

**An incorrect answer:** Intuitively, \( \mathcal{N} \) does, since besides even numbers, it also contains odd numbers.

**A correct one:** It is easy to see that \( f : \mathcal{N} \mapsto \mathcal{E}, \forall n \in \mathcal{N}, f(n) = 2n \) is both one-to-one: \( \forall n_1 \neq n_2, f(n_1) = 2n_1 \neq 2n_2 = f(n_2) \); and onto: \( \forall m = 2n \in \mathcal{E}, \) just pick \( n \in \mathcal{N} \), we have \( f(n) = m \).

Hence, by definition, \( |\mathcal{N}| = |\mathcal{E}| \).
Another example

Let \( Q \) be the set of all the rational numbers, \( \{\frac{m}{n} | m, n \in \mathbb{N}\} \).

**Question:** Which set contains more numbers?

Again, intuitively, \( |Q| > |\mathbb{N}| \), since the former contains all the fractions, besides the natural numbers.

But, once again, they are actually of the same size, because of the following correspondence.
Could they all have the same size?

**Definition:** A set is *countable* if either it is finite or it has the same size as $\mathcal{N}$.

**Question:** Is it true all the infinite sets are countable?

**A short answer:** No! In fact, if the answer were positive, cantor’s theory would not be interesting, at all.

**A longer one:** Let $\mathcal{C}$ be the set of all the real numbers, i.e., the numbers that have decimal representation, e.g., $3.1415926 \cdots$ etc. $|\mathcal{C}| > |\mathcal{N}|$. 
**Theorem:** $\mathcal{C}$ is not countable.

**Proof by contradiction:** Just assume that $\mathcal{C}$ is countable. By definition, there exists a one-to-one, onto function, $f$, such that it will map every natural number to a unique real number, and, for every real number, there is a corresponding natural number.

We construct a real number as follows: $x = 0.x_1x_2\cdots x_i\cdots$ such that for all $n \in \mathbb{N}, x_n \neq$ the $n^{\text{th}}$ digit of $f(n)$. As $x$ is obviously a real number, by definition, there must be a natural number $n_x$, such that $f(n_x) = x$. However, by construction, the $n_x^{\text{th}}$ digit of $x$ is not the same as the $n_x^{\text{th}}$ digit of $f(n_x) = x$. This is a contradiction.

Thus, there does not exist such a function $f$, i.e., $\mathcal{C}$ is not countable. \qed

**Homework:** 4.6-4.9.
Nothing is new under the Sun...

The key idea in the above proof uses the diagonal line of the list, thus the name. This technique is one of the few important ones that are frequently used in the study of the theory of computation.

The above result can be used to show that there exist languages that are not enumerable.

Since a TM is just a string, the set of Turing Machines is countable. But the set of all the languages, as a collection of all the subsets of strings, is not.

As each TM accepts only one language, there must be some language that is not accepted by any TM, thus, by definition, not enumerable.
Corollary: There are languages that are not enumerable.

Proof: To show that the set of TMs is countable, we merely observe that the set of all strings of $\Sigma^*$ for any finite alphabet $\Sigma$ is countable. With only finite number of strings of a given length, we can list all strings of length 0, followed by all strings of length 1, etc..

The set of all the TMs is countable, since any TM can be coded into a string. If we just cross over those strings that don’t represent a TM, we have a list of all the TMs.

To show that the set of languages is not countable, we observe that for any $\Sigma$, although $\Sigma^*$ is countable, $\mathcal{P}(\Sigma^*)$ is not, by establishing a one-to-one and onto correspondence with the set of all the infinite sequences consisting of 0 and 1. (?)
The main result

We now apply the diagonalization technique to show that $A_{TM}$ is not decidable. Namely, there does not exist a TM $H$, for any $\langle M, w \rangle$, $H$ is able to tell whether or not $M$ accepts $w$.

**Proof:** Just assume it is decidable and $H$ is its decider. Hence,

$$H(\langle M, w \rangle) = \begin{cases} 
\text{accept} & \text{if } M \text{ accepts } w, \\
\text{reject} & \text{otherwise.}
\end{cases}$$

Now, we use $H$ as a subroutine to construct a new TM, $D$, which determines what $M$ does when the input to $M$ is its own description, $\langle M \rangle$. 

23
What does $D$ do?

Once obtaining $\langle M \rangle$, the unary $D$ does the opposite of what the binary $H$ would with $\langle M, \langle M \rangle \rangle$.

For any input $\langle M \rangle$,

0. Use the copier that you constructed for the last chapter to construct $\langle M, \langle M \rangle \rangle$.

1. Run $H$ on input $\langle M, \langle M \rangle \rangle$.

2. If $H$ accepts $\langle M, \langle M \rangle \rangle$, i.e., $M$ accepts $\langle M \rangle$, rejects $\langle M \rangle$;

3. If $H$ rejects $\langle M, \langle M \rangle \rangle$, i.e., $M$ rejects $\langle M \rangle$, accepts $\langle M \rangle$.

Hence, we have that

$$D(\langle M \rangle) = \begin{cases} 
\text{accept} & \text{if } M \text{ rejects } \langle M \rangle, \\
\text{reject} & \text{otherwise.}
\end{cases}$$
The punch line...

*What happens when we run* $D$ *with its own description, $\langle D \rangle$?*

**Answer:** We must have

$$D(\langle D \rangle) = \begin{cases} 
\text{accept } \langle D \rangle & \text{if } D \text{ rejects } \langle D \rangle, \\
\text{reject } \langle D \rangle & \text{otherwise.}
\end{cases}$$

In other words,

$D$ accepts $\langle D \rangle$ if and only if $D$ rejects $\langle D \rangle$.

Thus, there doesn’t exist such a $D$, which implies that there could not exist such a decider $H$ for the original Halting problem, either.

This ends the proof of this undecidability result of $A_{TM}$.  \qed
Where is the beef?

The following assumes what $M_i$ will do with $\langle M_i \rangle$:

<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>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>Accept</td>
<td>Accept</td>
<td></td>
<td></td>
<td>\ldots</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>\ldots</td>
</tr>
<tr>
<td>$M_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\ldots</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Accept</td>
<td>Accept</td>
<td></td>
<td></td>
<td>\ldots</td>
</tr>
</tbody>
</table>

Assume that $H$ exists, the following gives the values of $H(M_i, \langle M_i \rangle)$:

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

Then what?
Here it is!

The following shows what $D$ would do and where the contradiction occurs:

<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>$\ldots$</th>
<th>$\langle D \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>Accept</td>
<td>reject</td>
<td>Accept</td>
<td>reject</td>
<td></td>
<td>Accept</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
<td>Accept</td>
</tr>
<tr>
<td>$M_3$</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Accept</td>
<td>Accept</td>
<td>reject</td>
<td>reject</td>
<td>Accept</td>
<td>Accept</td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>reject</td>
<td>reject</td>
<td>Accept</td>
<td>Accept</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By construction, $D$ rejects, e.g., $\langle M_1 \rangle$ since $M_1$ accepts $\langle M_1 \rangle$, but accepts, e.g., $\langle M_3 \rangle$ since $M_3$ rejects $\langle M_3 \rangle$, .... The question is what $D$ would do with $\langle D \rangle$?

Again by construction, $D$ should accept $\langle D \rangle$, if $D$ rejects $\langle D \rangle$; and reject $\langle D \rangle$, if $D$ accepts $\langle D \rangle$. In terms of the picture, we should fill in “Accept” in the ‘?’ if and only if it is filled with “reject”.

Thus, a contradiction occurs at “?”.  

27
Near the end

We showed that, since all the TMs are countable, but all the languages are not, at least one language is not associated with a TM. As a result, we concluded in Page 22 that some language is not TM enumerable.

We now show that $A_{TM}$ is one of such languages.

Recall that $\overline{A}$, the complement of a language $A$, is the collection of all the strings that don’t belong to $A$.

We call an enumerable language $\overline{E}$ co-enumerable if it is the complement of an enumerable language $E$. 
**Theorem:** A language is decidable if and only if it is both enumerable and co-enumerable.

**Proof:** As any decidable language is also enumerable and the complement of a decidable language is also decidable, the “only-if” part is done.

Let $M_1$ and $M_2$ be TMs that accept a language, $A$, and its complement, we construct the following TM for the “if” part. For any input $w$,

1. Run both $M_1$ and $M_2$ on input $w$ in the “dovetail” style. (Cf. Page 29 of the Computability Chapter notes)

2. If $M_1$ accepts $w$, accept; if $M_2$ accepts $w$, reject.

As $w$ either belongs to $A$ or $\overline{A}$, thus, either $M_1$ or $M_2$ will accept $w$. Thus, $M$ is a decider for $A$. 

29
Corollary: $\overline{A_{TM}}$ is not enumerable.

Proof: We know that $A_{TM}$ is enumerable. (Cf. Page 14)

If its complement were also enumerable, $A_{TM}$ would be decidable. This would contradict the main result, i.e., $A_{TM}$ is not decidable.

This provides a concrete example outside the Turing-recognizable bubble in Page 12.