Decidable Languages

- We use languages to represent various computational problems because we have a terminology for dealing with languages.
- **Definition:** A language is *decidable* if there is an algorithm (i.e., a Turing decider) to recognize it.
- We develop examples of languages that are decidable by algorithms.

**Theorem 4.1**

\[ A_{\text{DFA}} \] is a decidable language.

**Proof idea:** construct a TM \( M \) that decides \( A_{\text{DFA}} \)

\[ M = \{ \langle B, w \rangle \} \text{ where } B \text{ is a DFA and } w \text{ is a string} \]

1. Simulate \( B \) on \( w \)
2. If the simulation ends in an accept state then accept; if it ends in a nonaccepting state then reject.

**Note:** \( w \) is finite and simulation always ends.

Membership Problem for DFA

- Consider the acceptance problem for DFAs: test whether a particular finite automaton accepts a given string. This can be expressed as a language \( A_{\text{DFA}} \)
- \( A_{\text{DFA}} \) contains the encodings of all DFAs together with strings the DFAs accept, i.e., \( A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts } w \} \)
- Hence, testing whether DFA \( B \) accepts \( w \) is the same as testing whether \( \langle B, w \rangle \in A_{\text{DFA}} \)
- Other computational problems are formulated in terms of testing membership in a language.
- To show that a computational problem is decidable is to show that the encoding of the problem is decidable.
Acceptance Problem for NFA

\( A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA and } w \text{ is a string} \} \)

**Theorem** \( A_{NFA} \) is a decidable language

**Proof:** Construct a TM \( N \) that decides \( A_{NFA} \).

Because \( M \) is designed to work with DFAs, \( N \) first converts its input NFA to a DFA by the usual technique.

\( N = \) "On input \( \langle B, w \rangle \) where \( B \) is an NFA and \( w \) is a string

1. Convert NFA \( B \) to a DFA \( C \) (see Theorem 1.19)
2. Run TM \( M \) from Theorem 4.1 on \( \langle C, w \rangle \)
3. If \( M \) accepts, accept; otherwise reject"

**Note:** running \( M \) in stage 2 means incorporating \( M \) into the design of \( N \) as a subprocedure

---

Performing the Simulation

\( \langle B, w \rangle \) is a representation of a DFA \( B \) together with a string \( w \). One can represent \( B \) by a list of its five components: \( (Q, \Sigma, \delta, q_0, F) \)

- When \( M \) receives an input it checks first whether this input represents a DFA \( B \) and a string \( w \); if not reject
- If input is right, \( M \) keeps track of \( B \)’s current state and \( B \)’s current position in \( w \) by writing this info on its tape
- Initially the state of \( B \) is \( q_0 \) and \( B \)’s current position is the leftmost symbol of \( w \); the states and position are updated as shown by \( \delta \)
- When \( M \) finishes processing the last symbol of \( w \), \( M \) accepts if \( B \) is in a final state and reject if \( B \) is not in a final state

---

Theorem 4.4

\( E_{DFA} \) is a decidable language

**Proof idea:**

- A DFA accepts some string if reaching a final state from the start state by traveling along the arrows of the DFA is possible.
- To test this condition we can construct the TM \( T \) that marks states of DFA using the state transition function of the DFA
- Use \( T \) to solve emptiness problem

---

Emptiness Problem

Another kind of problems concerning FAs is the *emptiness testing*

**The problem:** test if the language of a DFA is empty

**For this case we consider the language**

\( E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \)
Language Equality

- The problem: for two DFAs $A$ and $B$, is $L(A) = L(B)$?
- The language: $EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs, and } L(A) = L(B) \}$

The TM $T$

$T$ = “On input $\langle A \rangle$ where $A$ is a DFA:
1. Mark the start state of $A$
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked
4. If no final state is marked, accept; otherwise reject”

Theorem 4.5

$EQ_{DFA}$ is a decidable language

Proof idea: (use Theorem 4.4)
- Construct a DFA $C$ from $A$ and $B$ where $C$ accepts only those strings that are accepted either by $A$ or $B$ but not by both.
- If $A$ and $B$ recognize the same language then $C$ accepts nothing
- The language $C$ is defined by $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$
- which is called symmetric difference of $L(A)$ and $L(B)$
- Use machine $T$ to check if $C$ is empty

Symmetric Difference

The expression $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ called the symmetric difference of $L(A)$ and $L(B)$ is illustrated below:

Figure 1: Symmetric difference of $L(A)$ and $L(B)$
Proving Theorem 4.5

Construct the TM \( F \): \( F = \langle A, B \rangle \) where \( A \) and \( B \) are DFAs.

1. Construct DFA \( C \) that recognizes \( L(C) \) as described above.
2. Run TM \( T \) from Theorem 4.4 on input \( \langle C \rangle \).
3. If \( T \) accepts, accept; if \( T \) rejects, reject.*

Solution

The TM \( L \) that decides \( \text{ALL}_{\text{DFA}} \) uses the fact that \( L(A) \) is regular.

1. Construct DFA \( B \) that recognizes \( L(A) \) by swapping accept and unaccept states in \( A \).
2. Run the TM \( T \) that decides the emptiness \( E_{\text{DFA}} \) on \( B \).
3. If \( T \) accepts, reject; if \( T \) rejects, accept.*

Decision Problems of CFLs

Describe algorithms to test whether a CFG generates a particular string.

Describe algorithms to test whether the language generated by a CFG is empty.

The language we consider is:

\[ \text{ALL}_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]
A better idea

- Make the recognizer a decider. For that we need to ensure that the algorithm tries only finitely many derivations.

Theorem 4.7

$A_{CFG}$ is a decidable language

Proof ideas: For a CFG $G$ and a string $w$:

- First idea: Go through all derivations generated by $G$ checking whether one of them is a derivation of $w$.

Since there are infinitely many derivations this idea does not work. If $G$ does not generate $w$ the algorithm doesn’t halt. I.e, this idea provides a recognizer but not a decider.

Checking CFG derivations

- Convert $G$ into Chomsky normal form
- For a string $w$ of length $|w| = n$ check all derivations with $2n - 1$ steps to determine whether $G$ generates $w$
- Since we can convert $G$ into a Chomsky normal form (see Section 2.1), this is a good idea

Fact

If $G$ is a CFG in Chomsky normal then for any $w \in L(G)$ where $|w| = n$ exactly $2n - 1$ steps are required for any derivation of $w$

Proof: Induction on $n$. 
Proof of Theorem 4.7

The TM $S$ that decide $A_{CFG}$ is:

$S = "On 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 $2n - 1$ steps, $n = \text{length}(w)$ except if $n = 0$; in this case list all derivations with 1 step
3. If any of the derivations listed above generates $w$, accept; if not reject."

Proof of Theorem 4.8

Construct the TM $R$:

$R = "On input \langle G \rangle where G is a CFG:
1. Mark all terminal symbols of $G$
2. Repeat until no new variable get marked:
   Mark any variable $A$ where $G$ has a rule $A \rightarrow u_1 u_2 \ldots u_k$ and each symbol $u_1, u_2, \ldots, u_k$ has already been marked
3. If the start symbol of $G$ is not marked, accept; otherwise reject."
Fact 1

Class of CF languages is not closed under intersection.

**Proof:** By construction.

- Consider the CF languages $A = \{a^m b^n c^n | m, n \geq 0\}$ and,
  $B = \{a^m b^n c^n | m, n \geq 0\}$ generated by the grammars:
  $S \rightarrow RT, R \rightarrow aR | \epsilon, T \rightarrow bTc | \epsilon$ and
  $S \rightarrow TR, T \rightarrow aTb | \epsilon, R \rightarrow cR | \epsilon$ respectively.
- $L(A) \cap L(B) = \{a^n b^n c^n | n \geq 0\}$ which is not a CFL
- This establishes Fact 1.

CFL equality problem

Consider the language:
$EQ_{CFG} = \{ \langle G, H \rangle | G \wedge H are CFG and L(G) = L(H) \}$

**Note:**
- Since class of CF languages is not closed under intersection and complementation (as seen before), we cannot use the symmetric difference for $EQ_{CFG}$.
- In fact $EQ_{CFG}$ is not decidable (to be proven later)

Fact 2

Class of CF languages is not closed under complementation.

**Proof:** By contradiction. Assume that CFL is closed under complementation
- If $G_1$ and $G_2$ are two CFG then $L(G_1)$ and $L(G_2)$ are CFL
- Then $L(G_1) \cup L(G_2)$ is a CFL. Hence, $L(G_1) \cup L(G_2)$ is a CFL.
- By DeMorgan's law $L(G_1) \cup L(G_2) = L(G_1) \cap L(G_2)$, a contradiction because class of CFL is not closed under intersections.

Theorem 4.9

Every context-free language is decidable.

**Proof:** Let $G$ be a CFG for $A$, i.e. $L(G) = A$. Design a TM $M_G$ that decides $A$ by building a copy of $G$ into $M_G$:

- $M_G = \langle\text{On input } w:\rangle$
  1. Run TM $S$ on input $\langle G, w \rangle$
  2. If this machine accepts, accept; if it rejects, reject

**Note:** TM $S$ converts $G$ to Chomsky normal form, and produces all derivations of length $2n - 1$ where $n = |w|$. Then check if $w$ is among the derived strings.
Example 1

Equivalence of DFA and REX:
- Consider the problem of testing whether a DFA and a regular expression are equivalent.
- Express this problem as a language and show that this language is decidable.

Example 2

Decidability of $\Sigma^*$
- Problem: is the language $\Sigma^*$, for $\Sigma$ a finite alphabet, decidable?
- Language: $ALL_{DFA} = \{ \langle A \rangle \mid A$ is a DFA that recognize $\Sigma^* \}$ Show that $ALL_{DFA}$ is decidable.

Methodology

To solve decidability problems concerning relations between languages one should proceed as follows:
- Understand the relationship
- Transform the relationship into an expression using closure operators on decidable languages
- Design a TM that construct the language thus expressed
- Run a TM that decide the language represented by the expression

Solution

- Let $EQ_{DFA, REX} = \{ \langle A, R \rangle \mid A$ is a DFA, $R$ is a regular expression and $L(A) = L(R) \}$
- The following TM $E$ decides $EQ_{DFA, REX}$:
  $E = "On$ input $\langle A, R \rangle$
  1. Convert $R$ to an equivalent DFA $B$
  2. Use the TM $C$ for deciding $EQ_{DFA}$ on input $\langle A, B \rangle$
  3. If $C$ accepts, accept; if $C$ rejects, reject."

Note: $C$ constructs the symmetric difference $(L(A) \cap L(B)) \cup (\overline{L(A)} \cap L(B))$ and test if it is empty.
Example 3

Using CFG and REX

- **Problem**: show that the problem of testing whether a CFG generates some string in $1^*$ is decidable.
- **Language**: $A = \{(G) \mid G \text{ is a CFG over } \{0,1\}^* \text{ and } 1^* \cap L(G) \neq \emptyset\}$

Solution

The TM $L$ that decides $ALL_{DFA}$ uses the fact that $\overline{L(A)}$ is regular

$L = "On input \langle A \rangle \text{ where } A \text{ is a DFA:}"$

1. Construct DFA $B$ that recognizes $\overline{L(A)}$ by swapping accept and unaccept states in $A$
2. Run the TM $T$ that decides the emptiness $E_{DFA}$ on $B$
3. If $T$ accepts, **reject**; if $T$ rejects **accept**.

Example 4

Example regular expressions

- **Problem**: Is the language generated by a particular regular expression decidable? For example, is the language of regular expressions that contain at least one string that has the pattern “111” as a substring decidable?
- **Language**: $A = \{(R) \mid R \text{ is a regular expression describing a language that contain at least one string } w \text{ that has “111” as a substring (i.e., } w = x111y \text{ where } x \text{ and } y \text{ are strings)}$}

Solution

Assume that $G$ is over $\{0,1\}^*$. Then we need to show that the language $A = \{(G) \mid G \text{ is a CFG over } \{0,1\}^* \text{ and } 1^* \cap L(G) \neq \emptyset\}$ is decidable. Since $1^*$ is regular and $L(G)$ is CFL then $1^* \cap L(G)$ is a CFL. Hence the TM $X$ that decides $A$ is:

$X = "On input \langle G \rangle \text{ where } G \text{ is a CFG:}"$

1. Construct CFG $H$ such that $L(H) = 1^* \cap L(G)$
2. Run the TM $R$ that decides the language $E_{CFG}$ on $\langle H \rangle$
3. If $R$ accepts, **reject**; if $R$ rejects, **accept**.
**Halting Problem of TM**

- It is also the membership problem of TM: whether a Turing machine accepts a given input string.
- By analogy with $A_{DFA}$ and $A_{CFG}$ we call the corresponding language $A_{TM}$
  
  $$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$$

- **Solution**

  The language $A$ is decidable. The reason is that language $A$ can be expressed using regular operators as $L(\Sigma^*111\Sigma^*) \cap L(R)$. Hence, the TM $X$ that decides $A$ is:

  **X** = "On input $\langle R \rangle$ where $R$ is a regular expression:

  1. Construct the DFA $E$ that accepts $\Sigma^*111\Sigma^*$
  2. Construct the DFA $B$ that accepts $L(B) = L(R) \cap L(E)$
  3. Run TM $T$ that decide $E_{DFA}$ on input $\langle B \rangle$
  4. If $T$ accepts reject, if $T$ rejects accept."

  **A recognizer for $A_{TM}$**

  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."

  **Theorem 4.11**

  $A_{TM}$ is undecidable.

  - $A_{TM}$ is however Turing-recognizable
  - Hence, Theorem 4.11 shows that recognizers are more powerful than deciders.
  - Requiring a TM to halt on all inputs restricts the kind of languages that it can recognize
**Observations**

- The TM $U$ is interesting in its own right because it is an example of the *universal Turing machine*, first proposed by Turing.
- $U$ is called universal because it is capable to simulate any other Turing machine from the description of that machine.
- The universal TM played an important role in the stimulating the development of stored-program computers.

**Note**

- $U$ loops on the input $\langle M, w \rangle$ if $M$ loops on $w$. This is why $U$ does not decide $A_{TM}$.
- If the algorithm had some way to determine that $M$ was not halting on $w$, it could reject. This is why it is called the halting problem.
- However, Theorem 4.11 states that an algorithm has no way to make this determination.

**Example Infinite Sets**

- The set of strings over $\{0, 1\}$ is an infinite set.
- The set $\mathcal{N}$ of natural numbers is also an infinite set.
- The set $\mathcal{E}$ of all even natural numbers is also an infinite set.
- How can we compare them?

**How to Prove Undecidability**

- The proof of undecidability of the TM membership problem uses Georg Cantor (1873) technique called *diagonalization*.
- Cantor's problem was to measure the size of infinite sets.
- The size of finite sets is measured by counting the number of their elements.
- The size of infinite sets cannot be measured by counting their elements because this procedure does not halt.
Two finite sets have the same size if their elements can be paired. Since this method does not rely on counting elements it can be used for both finite and infinite sets.

Consider two sets, \( A \) and \( B \) and \( f : A \to B \) a function.

- If \( f \) is one-to-one if it never maps two different elements of \( A \) into the same element of \( B \), i.e., \( f(a_1) \neq f(a_2) \) whenever \( a_1 \neq a_2 \).
- \( f \) is onto if it hits every element of \( B \), i.e., \( \forall b \in B, \exists a \in A \) such that \( f(a) = b \).

A correspondence \( f : A \to B \) is called a correspondence if it is both one-to-one and onto.

Two sets \( A \) and \( B \) have the same size if there is a correspondence \( f : A \to B \).

**Cantor's solution**

Let \( N \) be the set of natural numbers, \( N' = \{1, 2, 3, \ldots\} \) and \( \mathcal{E} \) the set of even natural numbers, \( \mathcal{E} = \{2, 4, 6, \ldots\} \). Intuitively one may believe that \( \text{size}(N') > \text{size}(\mathcal{E}) \) because \( \mathcal{E} \subset N' \) but the opposite. However, using Cantor method we can show that \( N' \) and \( \mathcal{E} \) have the same size by constructing the correspondence \( f : N' \to \mathcal{E} \) defined by \( f(n) = 2n \).

**Definition 4.14**

A set is countable if either it is finite or it has the same size as \( N' \).

**Example Correspondences**

Let \( N = \{1, 2, 3, \ldots\} \) be the set of natural numbers, \( N' = \{1, 2, 3, \ldots\} \) and \( \mathcal{E} \) the set of even natural numbers, \( \mathcal{E} = \{2, 4, 6, \ldots\} \). Intuitively one may believe that \( \text{size}(N') > \text{size}(\mathcal{E}) \) because \( \mathcal{E} \subset N' \) but the opposite. However, using Cantor method we can show that \( N' \) and \( \mathcal{E} \) have the same size by constructing the correspondence \( f : N' \to \mathcal{E} \) defined by \( f(n) = 2n \).
**Correspondence \( \mathcal{Q} \leftrightarrow \mathcal{N} \)**

1. Put \( \mathcal{N} \) on two axes
2. Line \( i \) contains all rationals that have numerator \( i \), i.e.
   \[ \left\{ \frac{j}{i} \in \mathcal{Q} \mid i \in \mathcal{N}, j \in N \text{ fixed} \right\} \]
3. Column \( j \) contains all rationals that have denominator \( j \), i.e.
   \[ \left\{ \frac{i}{j} \in \mathcal{Q} \mid \forall i \in \mathcal{N}, j \in \mathcal{N} \text{ fixed} \right\} \]
4. Number \( \frac{i}{j} \) occurs in \( i \)-th row and \( j \)-th column

**A complex correspondence**

Let \( \mathcal{Q} \) be the set of positive rational numbers,
\[ \mathcal{Q} = \left\{ \frac{m}{n} \mid m, n \in \mathcal{N} \right\} \]
- Intuitively, \( \mathcal{Q} \) seems to be much larger than \( \mathcal{N} \)
- Yet we can show that this two sets have the same size by constructing a correspondence.

**The list of rational numbers**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{1}{1} )</td>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{3} )</td>
<td>( \frac{1}{4} )</td>
<td>( \frac{1}{5} )</td>
<td>...</td>
</tr>
<tr>
<td>( \frac{2}{1} )</td>
<td>( \frac{2}{2} )</td>
<td>( \frac{2}{3} )</td>
<td>( \frac{2}{4} )</td>
<td>( \frac{2}{5} )</td>
<td>...</td>
</tr>
<tr>
<td>( \frac{3}{1} )</td>
<td>( \frac{3}{2} )</td>
<td>( \frac{3}{3} )</td>
<td>( \frac{3}{4} )</td>
<td>( \frac{3}{5} )</td>
<td>...</td>
</tr>
<tr>
<td>( \frac{4}{1} )</td>
<td>( \frac{4}{2} )</td>
<td>( \frac{4}{3} )</td>
<td>( \frac{4}{4} )</td>
<td>( \frac{4}{5} )</td>
<td>...</td>
</tr>
<tr>
<td>( \frac{5}{1} )</td>
<td>( \frac{5}{2} )</td>
<td>( \frac{5}{3} )</td>
<td>( \frac{5}{4} )</td>
<td>( \frac{5}{5} )</td>
<td>...</td>
</tr>
</tbody>
</table>

**Turning \( \{ \frac{i}{j} \mid i, j \in \mathcal{N} \} \) into a list**

- Bad idea: list first elements of a line or a column. Lines and columns are labeled by \( \mathcal{N} \), hence this would never end
- Good idea (Cantor’s idea): use the diagonals:
  1. First diagonal contains \( \frac{1}{1} \), i.e. first element of the list is \( \frac{1}{1} \)
  2. Continue the list with the elements of the next diagonal: \( \frac{1}{2}, \frac{2}{1} \)
  3. Continue this way skipping the elements that may generate repetitions, such \( \frac{1}{1} \) that would generate a copy of \( \frac{1}{1} \).
Uncountable sets

An infinite set for which no correspondence with \( \mathbb{N} \) can be established is called **uncountable**

**Example of uncountable set:** the set \( \mathbb{R} \) of real numbers is uncountable

**Proof:** Cantor proved that \( \mathbb{R} \) is uncountable using the diagonalization method.

**Construction**

- Since \( f : \mathbb{N} \rightarrow \mathbb{R} \) is a correspondence \( \mathbb{R} \) can be listed

<table>
<thead>
<tr>
<th>( n )</th>
<th>( f(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.14159...</td>
</tr>
<tr>
<td>2</td>
<td>55.5555...</td>
</tr>
<tr>
<td>3</td>
<td>0.1234...</td>
</tr>
<tr>
<td>4</td>
<td>0.5000...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 3: Listing \( \mathbb{R} \)

More Infinite Countable Sets

- The set \( \mathbb{N} \times \mathbb{N} \)
- The set \( \mathbb{N}^k \) for any \( k \in \mathbb{N} \)
- The set \( \Sigma^* \)
- Any subset of a countable set is also countable.

**Theorem 4.17**

\( \mathbb{R} \) is uncountable.

**Proof:** We will show that no correspondence exist between \( \mathbb{N} \) and \( \mathbb{R} \).

- Suppose that such a correspondence \( f : \mathbb{N} \rightarrow \mathbb{R} \) exits and deduce a contradiction showing that \( f \) fail to work properly.
- We construct an \( x \in \mathbb{R} \) that cannot be the image of any \( n \in \mathbb{N} \).
Application

Theorem 4.17 shows that some languages are not decidable or even Turing recognizable.

Reason:
- There are uncountably many languages yet only countably many Turing machines.
- Because each Turing machine can recognize a single language and there are more languages than Turing machines some languages are not recognized by any Turing machine.
- Such languages are not Turing recognizable.

Formal construction of $x$

Construct $x \in (0,1)$ by the following procedure:

$x = 0.d_1d_2d_3d_4\ldots$ has an infinite number of decimals constructed by the rule:

$\forall i \in \mathbb{N}$ chose $d_i$ a digit different from the $i$-th digit of $f(i)$

Consequence: $\forall i \in \mathbb{N}$, $x \neq f(i)$. Hence, $x$ does not belong to the list $\mathcal{R}$ and thus $f$ is not a correspondence.

Uncountable Sets

- The real numbers $\mathcal{R}_1 = (0,1)$
- The infinite-length binary strings $\mathcal{B}$
- The integer functions $\mathcal{F} = \{ f \mid f : \mathbb{N} \to \{0,1\} \}$
- The powerset $\mathcal{P}(\mathbb{N})$
- The set of all formal languages $\mathcal{L} = \{ L \mid L \subseteq \Sigma^* \}$

Countable Sets

- Any finite set
- Natural numbers $\mathbb{N}$
- Rational numbers $\mathbb{Q}$
- Any subset of a countable set
- Cartesian product of countable sets
- $\Sigma^*$
A characteristic function

Let \( \Sigma = \{0, 1\} \) and \( A \) the language of all strings starting with 0 over \( \Sigma \).

\[
\begin{align*}
\Sigma &= \{\epsilon, 0, 1, 00, 01, 11, 000, 001, 010, 011, 100, \ldots\}; \\
A &= \{0, 00, 01, 000, 001, 010, 011, \ldots\}; \\
\chi_A &= 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ \ldots
\end{align*}
\]

Corollary 4.18

Some languages are not Turing-recognizable.

Proof idea:
- The set of all languages is uncountable.
- The set of all Turing machines is countable.

Halting Problem is Undecidable

We are ready to prove that the language

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

is undecidable.

Proof Idea: By contradiction, assuming that \( A_{TM} \) is decidable.

Suppose that \( H \) is a decider of \( A_{TM} \).

On input \( \langle M, w \rangle \) where \( M \) is a TM and \( w \) is a string, \( H \) halts and accepts if \( M \) accepts \( w \).

Furthermore, \( H \) halts and reject if \( M \) fails to accept \( w \).
Proof, continuation

Construct a new TM $D$ that uses $H$ as a subroutine.

- $D$ calls $H$ to determine what $M$ does when its input is $w = \langle M \rangle$
- If $M$ accepts $\langle M \rangle$ then $D$ rejects;
  if $M$ rejects $\langle M \rangle$ then $D$ accepts

Equational expression of $H$

$$H(\langle M, w \rangle) = \begin{cases} 
  \text{accept}, & \text{if } M \text{ accepts } w; \\
  \text{reject}, & \text{if } M \text{ does not accept } w.
\end{cases}$$

A CONTRADICTION

$$D(\langle M \rangle) = \begin{cases} 
  \text{accept}, & \text{if } M \text{ does not accept } \langle M \rangle; \\
  \text{reject}, & \text{if } M \text{ accepts } \langle M \rangle.
\end{cases}$$

What happens when we ran $D$ on $\langle D \rangle$?

$$D(\langle D \rangle) = \begin{cases} 
  \text{accept}, & \text{if } D \text{ does not accept } \langle D \rangle; \\
  \text{reject}, & \text{if } D \text{ does not reject } \langle D \rangle.
\end{cases}$$

This is a contradiction and consequently neither TM $D$ nor TM $H$ do exist.

The machine $D$

$D$ = "On input $\langle M \rangle$, where $M$ is a TM:
1. Run $H$ on input $\langle M, \langle M \rangle \rangle$
2. Output the opposite of what $H$ outputs: accepts if $H$ rejects and rejects if $H$ accepts."
Where is diagonalization?

To make the use of diagonalization obvious we construct the list of all Turing machines running on Turing machines as input:

<table>
<thead>
<tr>
<th></th>
<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>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_2</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>M_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>M_4</td>
<td>accept</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Entry (i,j) is accept if M_i accepts (M_j)

Running D on (D)

The result of running H when D is present.

<table>
<thead>
<tr>
<th></th>
<th>(M_1)</th>
<th>(M_2)</th>
<th>(M_3)</th>
<th>(M_4)</th>
<th>...</th>
<th>(D)</th>
<th>...</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>
<td>...</td>
</tr>
<tr>
<td>M_2</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>...</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>M_3</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>...</td>
<td>reject</td>
<td>...</td>
</tr>
<tr>
<td>M_4</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>...</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>D</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>...</td>
<td>???</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 6: A contradiction occurs at (D, (D))

Running H

The result of running H:

<table>
<thead>
<tr>
<th></th>
<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>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>...</td>
</tr>
<tr>
<td>M_2</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>M_3</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>M_4</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 5: Entry (i,j) is the value of H on (M_i, (M_j))

Summarizing

- Assume that H decides $A_{TM}$
- Use H to build D that accepts $⟨M⟩$ when M rejects and rejects $⟨M⟩$ when M accepts
- H and D performs as follows:
  - H accepts $⟨M, w⟩$ exactly when M accepts w
  - D rejects $⟨M⟩$ exactly when M accepts $⟨M⟩$
  - D rejects $⟨D⟩$ exactly when D accepts $⟨D⟩$

This is a contradiction and neither H nor D can exist.

Theory of Computation – p.69/
A New Concept

Co-Turing recognizable languages
- Complement of a language $A$, $C(A)$, is the language consisting of all strings that does not belong to $A$.
- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language

Proof

if Assume that $A$ is decidable. Since complement of a decidable language is decidable it result that both $A$ and $C(A)$ are Turing-recognizable.

only if Assume that both $A$ and $C(A)$ are Turing-recognizable. Let $M_1$ be a recognizer for $A$ and $M_2$ a recognizer for $C(A)$. Then the following TM $M$ is a decider for $A$

Note

We can construct a Turing-unrecognizable language
- $A_{TM}$ is an example of Turing undecidable language. But it is Turing recognizable
- Now we construct a language which is Turing-unrecognizable.
- This construction relies on the fact that if both a language and its complement are Turing-recognizable the language is decidable
- Hence: for any undecidable language, either the language or its complement is not Turing-recognizable

Theorem 4.16

A language is decidable iff it is both Turing-recognizable and co-Turing recognizable
I.e., a language $A$ is decidable iff both $A$ and $C(A)$ are Turing-recognizable
Note

- Running two machines $M_1$ and $M_2$ by a machine $M$ in parallel means that $M$ has two tapes, one for simulating $M_1$ and other for simulating $M_2$.
- $M$ takes turns, simulating one step of each machine, which continues until one of the machines halts.
- Because $w \in A$ or $w \in C(A)$ either $M_1$ or $M_2$ must accept $w$.
- Because $M$ halts whenever $M_1$ or $M_2$ accepts, $M$ always halts, so it is a decider. Further, it accepts all strings from $A$ and rejects all strings not in $A$.

Construction

$M =$ "On input $w$:
1. Run both $M_1$ and $M_2$ on $w$ in parallel
2. If $M_1$ accepts $w$ accept; if $M_2$ accepts $w$ reject."

Corollary

$C(A_{TM})$ is not Turing-recognizable

Proof: We know that $A_{TM}$ is Turing-recognizable. If $C(A_{TM})$ also were Turing-recognizable then $A_{TM}$ would be decidable. But we have proved (Theorem 4.11) that $A_{TM}$ is not decidable. Hence, $C(A_{TM})$ must not be Turing-recognizable.

Conclusion

$M$ is a decider for $A$, thus $A$ is decidable.