CSCI 2670 Introduction to Theory of Computing

Lecture Note 4

**Computability Theory**: Chapter 3 and Chapter 4

November 28, 2018
Part Two: Computability Theory

We will answer questions:

• How to recognize languages like $\{a^n b^n c^n : n \geq 0\}$?

• More difficult languages and models that can compute them?

• Are there “not computable” languages?
Part Two: Computability Theory

We will answer questions:

• How to recognize languages like \{a^n b^n c^n : n \geq 0\}?

• More difficult languages and models that can compute them?

• Are there “not computable” languages?
Part Two: Computability Theory

We will answer questions:

- How to recognize languages like \( \{ a^n b^n c^n : n \geq 0 \} \) ?
Part Two: Computability Theory

We will answer questions:

• How to recognize languages like $\{a^n b^n c^n : n \geq 0\}$?

• More difficult languages and models that can compute them?
Part Two: Computability Theory

We will answer questions:

• How to recognize languages like \( \{a^n b^n c^n : n \geq 0\} \) ?
• More difficult languages and models that can compute them ?
• Are there “not computable” languages ?
Chapter 3. Church-Turing Thesis
Chapter 3. Church-Turing Thesis

- 3.1 Turing Machines (TMs)
  Formal definition of the most basic TM model, examples;
Chapter 3. Church-Turing Thesis

- 3.1 Turing Machines (TMs)
  Formal definition of the most basic TM model, examples;

- 3.2 Variant of TMs
  Extended models but all equivalent to the basic TM

The first problem that does not have algorithm to solve (Hilbert’s 10th problem)
Chapter 3. Church-Turing Thesis

- 3.1 Turing Machines (TMs)
  Formal definition of the most basic TM model, examples;
- 3.2 Variant of TMs
  Extended models but all equivalent to the basic TM
- 3.3 Definition of Algorithms
  Church-Turing thesis (basic TM mode is the most powerful computing machine)
Chapter 3. Church-Turing Thesis

- 3.1 Turing Machines (TMs)
  Formal definition of the most basic TM model, examples;

- 3.2 Variant of TMs
  Extended models but all equivalent to the basic TM

- 3.3 Definition of Algorithms
  Church-Turing thesis (basic TM mode is the most powerful computing machine)
  The first problem that does not have algorithm to solve (Hilbert's 10th problem)
Chapter 4. Decidability
Chapter 4. Decidability

4.1 Decidable Languages

- **How to prove a language is decidable**
- Various decidable languages concerning computation models;
Chapter 4. Decidability

4.1 Decidable Languages

- How to prove a language is decidable
- Various decidable languages concerning computation models;

4.2 Undecidable Languages

- How to prove a language is not decidable
- The first language that was proved undecidable: Halting Problem
- Other undecidable languages
3.1 Turing Machines

A Turing machine (TM) consists of

1. an infinite input tape, where the input is placed,
2. a read head moving left and right on input and beyond, and can read and write on the input tape,
3. a set of states and transitions, and
4. accept and reject states, for the machine to halt.
3.1 Turing Machines

3.1 Turing Machines

A Turing machine (TM) consists of:

1. An infinite input tape, where the input is placed,
2. A read head moving left and right on input and beyond, and can read and write on the input tape,
3. A set of states and transitions, and
4. Accept and reject states, for the machine to halt.
3.1 Turing Machines

3.1 Turing Machines

A *Turing machine* (TM) consists of

1. an infinite *input tape*, where the input is placed,
3.1 Turing Machines

A Turing machine (TM) consists of

1. an infinite input tape, where the input is placed,
2. a read head moving left and right on input and beyond,
A Turing machine (TM) consists of

1. an infinite input tape, where the input is placed,
2. a read head moving left and right on input and beyond, and can read and write on the input tape,
3.1 Turing Machines

A Turing machine (TM) consists of

(1) an infinite input tape, where the input is placed,
(2) a read head moving left and right on input and beyond, and can read and write on the input tape,
(3) a set of states and transitions, and
3.1 Turing Machines

A *Turing machine* (TM) consists of

1. an infinite *input tape*, where the input is placed,
2. a *read head* moving left and right on input and beyond, and can *read and write* on the input tape,
3. a set of states and transitions, and
4. *accept* and *reject* states, for the machine to *halt*. 
3.1 Turing Machines

A Turing machine (TM) consists of

(1) an infinite input tape, where the input is placed,
(2) a read head moving left and right on input and beyond, and can read and write on the input tape,
(3) a set of states and transitions, and
(4) accept and reject states, for the machine to halt.
3.1 Turing Machines

Notes:
• A TM has "random access memory", more powerful than a PDA.
  e.g., recognizing $L = \{w \# w : w \in \{0, 1\}^*\}$
3.1 Turing Machines

Notes:

• A TM has “random access memory”, more powerful than a PDA.
3.1 Turing Machines

Notes:

- A TM has “random access memory”, more powerful than a PDA.
  
e.g., recognizing \( L = \{w\#w : w \in \{0, 1\}^*\} \)
3.1 Turing Machines

Notes:

- A TM has “random access memory”, more powerful than a PDA.
  e.g., recognizing $L = \{w\#w : w \in \{0, 1\}^*\}$
3.1 Turing Machines

A TM may not halt, e.g., keep moving to the right on the infinite tape. There are 3 possible outcomes of the TM computation:

1. accept and halt;
2. reject and halt;
3. not halt;
A TM may not halt,
3.1 Turing Machines

- A TM may not halt,
  
e.g., keep moving to the right on the infinite tape.
3.1 Turing Machines

- A TM may not halt,
  e.g., keep moving to the right on the infinite tape.
  There are 3 possible outcomes of the TM computation:
3.1 Turing Machines

- A TM may not halt,
  e.g., keep moving to the right on the infinite tape.
  There are 3 possible outcomes of the TM computation:
    (1) accept and halt; (2) reject and halt; (3) not halt;
3.1 Turing Machines

Definition 3.3
A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where
1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet, not including the blank symbol $\mathbb{0}$,
3. $\Gamma$ is the tape alphabet, where $\mathbb{0} \in \Gamma$ 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, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$. - about input tape: when it starts, the input tape has content $x_1x_2...x_n$, and the rest of the tape consists of blank symbols $\mathbb{0}$'s.

- about transition function $\delta$: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ cannot move off the leftmost position, even if $L$ is used.
3.1 Turing Machines

Formal definition of a Turing machine
Definition 3.3
3.1 Turing Machines

Formal definition of a Turing machine
Definition 3.3

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where $Q, \Sigma, \Gamma$ are all finite sets, and
3.1 Turing Machines

Formal definition of a Turing machine
Definition 3.3

A *Turing machine* is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,

...
3.1 Turing Machines

Formal definition of a Turing machine
Definition 3.3

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the blank symbol \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) 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, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{accept}} \neq q_{\text{reject}}\).
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the blank symbol \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta:\ Q \times \Gamma \to Q \times \Gamma \times \{L, R\}\) is the transition function,
5. \(q_0 \in Q\) is the start state,
6. \(q_{accept} \in Q\) is the accept state, and
7. \(q_{reject} \in Q\) is the reject state, where \(q_{accept} \neq q_{reject}\).
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the blank symbol \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\) is the transition function,
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the blank symbol \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) 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,
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A *Turing machine* is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the *blank symbol* \(\square\),
3. \(\Gamma\) is the tape alphabet, where \(\square \in \Gamma\) 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, and
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A *Turing machine* is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the *blank symbol* \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) 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. \(q_{accept} \in Q\) is the accept state, and
7. \(q_{reject} \in Q\) is the reject state, where \(q_{accept} \neq q_{reject}\).
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the blank symbol \(\square\),
3. \(\Gamma\) is the tape alphabet, where \(\square \in \Gamma\) 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, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{accept}} \neq q_{\text{reject}}\).

- about input tape:
  when it starts, the input tape has content \(x_1 x_2 \ldots x_n\), and
  the rest of the tape consists of blank symbols \(\square\)’s
3.1 Turing Machines

Formal definition of a Turing machine

Definition 3.3

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet, not including the blank symbol \(\square\),
3. \(\Gamma\) is the tape alphabet, where \(\square \in \Gamma\) 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, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{accept}} \neq q_{\text{reject}}\).

- about input tape:
  when it starts, the input tape has content \(x_1 x_2 \ldots x_n\), and the rest of the tape consists of blank symbols \(\square\)’s

- about transition function \(\delta\):
  \(\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\)
  cannot move off the leftmost position, even if \(L\) is used.
3.1 Turing Machines

A simple example of transition function for a Turing machine:

\[ \delta(q_0, 0) = (q_0, 0, R) \]

\[ \delta(q_0, 1) = (q_{\text{accept}}, 0, R) \]

\[ \delta(q_0, \_\_\_) = (q_{\text{reject}}, 0, R) \]

Transitions described as:

\[ a \rightarrow b, L/R \]

\[ L = \{ w : w \text{ contains symbol 1} \} \]
3.1 Turing Machines

A simple example of transition function for a Turing machine

\[ \delta(q_0, 0) = (q_0, 0, R) \]

\[ \delta(q_0, 1) = (q_{\text{accept}}, 0, R) \]

\[ \delta(q_0, \text{\textual}) = (q_{\text{reject}}, 0, R) \]

transitions described with:

\[ a \rightarrow b, L/R \]

\[ L = \{ w : w \text{ contains symbol } 1 \} \]
3.1 Turing Machines

A simple example of transition function for a Turing machine

\[ \delta(q_0, 0) = (q_0, 0, R) \]
\[ \delta(q_0, 1) = (q_{accept}, 0, R) \]
\[ \delta(q_0, \sqcup) = (q_{reject}, 0, R) \]
3.1 Turing Machines

A simple example of transition function for a Turing machine

\[ \delta(q_0, 0) = (q_0, 0, R) \]
\[ \delta(q_0, 1) = (q_{\text{accept}}, 0, R) \]
\[ \delta(q_0, \sqcup) = (q_{\text{reject}}, 0, R) \]
A simple example of transition function for a Turing machine

\[ \delta(q_0, 0) = (q_0, 0, R) \]
\[ \delta(q_0, 1) = (q_{accept}, 0, R) \]
\[ \delta(q_0, \sqcup) = (q_{reject}, 0, R) \]

transitions described with: \( a \rightarrow b, L/R \)

\( L = \{ w : w \text{ contains symbol 1 } \} \)
3.1 Turing Machines

Can you modify the transition function for a more complicated task? e.g., checking if the last digit is 1.
- scanning to the right until seeing ⊔
- checking the digit on the left.
3.1 Turing Machines

Can you modify the transition function for a more complicated task? e.g., checking if the last digit is 1.
- scanning to the right until seeing ⊔
- checking the digit on the left.
3.1 Turing Machines

Can you modify the transition function for a more complicated task?

- e.g., checking if the last digit is 1.
  - scanning to the right until seeing ⊓ and
  - checking the digit on the left.
3.1 Turing Machines

The notion of configuration to define the machine status at any moment is:
1. current state,
2. current read head position,
3. current tape content,

For example, configuration 1 0 1 1 q 7 0 1 1 1 1, assume \( \Sigma = \{0, 1\} \), initial configuration, assume \( \Sigma = \{a, b, c\} \), \( \Gamma = \Sigma \cup \{\# , \$, & \} \).

A few steps later:

\[
\begin{array}{cccc}
# & # & q & 4 \\
a & a & $ & $ \\
& b & b & & \\
& & & & c & c\end{array}
\]

Note that the read head points to the symbol to the right of the state symbol.
3.1 Turing Machines

The notion of configuration

The current state, current read head position, and current tape content define the machine status at any moment.
3.1 Turing Machines

The notion of configuration to define the machine status at any moment

(1) current state,
(2) current read head position,
(3) current tape content,
3.1 Turing Machines

The notion of **configuration**

to define the machine status at any moment

(1) current state,
(2) current read head position,
(3) current tape content,

e.g., configuration 1 0 1 1q70 1 1 1 1, assume Σ = \{0, 1\},
The notion of configuration to define the machine status at any moment:

1. current state,
2. current read head position,
3. current tape content,

For example, configuration $1 \ 0 \ 1 \ 1 q_7 \ 0 \ 1 \ 1 \ 1 \ 1$, assume $\Sigma = \{0, 1\}$.

For example, initial configuration, assume $\Sigma = \{a, b, c\}$, $\Gamma = \Sigma \cup \{\#, \$, \&, \}$

$q_0a \ a \ a \ a \ b \ b \ b \ b \ c \ c \ c \ c$
3.1 Turing Machines

The notion of configuration to define the machine status at any moment

(1) current state,
(2) current read head position,
(3) current tape content,

e.g., configuration 1 0 1 1q70 1 1 1 1, assume \( \Sigma = \{0, 1\} \),

e.g., initial configuration, assume \( \Sigma = \{a, b, c\} \), \( \Gamma = \Sigma \cup \{\#, $, \&, \sqcup\} \)

\[ q_0a \ a \ a \ a \ b \ b \ b \ b \ c \ c \ c \ c \]

a few steps later:

\[ # \ #q_4a \ a \ $ \ $ \ b \ b \ \& \ \& \ c \ c \]

Note that the read head points to the symbol to the right of the state symbol.
3.1 Turing Machines

Formalizing Turing machine computation:
A configuration $C_1$ yields configuration $C_2$ if the machine can go from $C_1$ to $C_2$ in a single step. Formally, let $a,b,c \in \Gamma$, $u,v \in \Gamma^*$, and $q_i,q_j \in Q$. We say $uaq_i bv$ yields $uacq_j v$ if
$$\delta(q_i,b) = (q_j,c,L)$$
and
$$\delta(q_i,b) = (q_j,c,R).$$
3.1 Turing Machines

Formalizing Turing machine computation:
3.1 Turing Machines

Formalizing Turing machine computation:

Configuration $C_1$ \textit{yields} configuration $C_2$
3.1 Turing Machines

Formalizing Turing machine computation:

Configuration $C_1$ yields configuration $C_2$

if the machine can go from $C_1$ to $C_2$ in a single step.
3.1 Turing Machines

Formalizing Turing machine computation:

Configuration $C_1$ yields configuration $C_2$

if the machine can go from $C_1$ to $C_2$ in a single step.

Formally, let $a, b, c \in \Gamma, u, v \in \Gamma^*$, and $q_i, q_j \in Q$. We say
3.1 Turing Machines

Formalizing Turing machine computation:

Configuration $C_1$ yields configuration $C_2$

if the machine can go from $C_1$ to $C_2$ in a single step.

Formally, let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, and $q_i, q_j \in Q$. We say

$$uaq_ibv \text{ yields } uq_jacv$$

if $\delta(q_i, b) = (q_j, c, L)$, and
3.1 Turing Machines

Formalizing Turing machine computation:

Configuration $C_1$ yields configuration $C_2$ if the machine can go from $C_1$ to $C_2$ in a single step.

Formally, let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, and $q_i, q_j \in Q$. We say

$$uaq_ibv \text{ yields } uq_jacv$$

if $\delta(q_i, b) = (q_j, c, L)$, and

$$uaq_ibv \text{ yields } uacq_jv$$

if $\delta(q_i, b) = (q_j, c, R)$. 
3.1 Turing Machines

- The start configuration is $q_0 w$ where $w$ is the input.
- A configuration is an accepting configuration if its state is the accepting state.
- A configuration is a rejecting configuration if its state is the rejecting state.
- Both accepting and rejecting configurations are halting configurations.
3.1 Turing Machines

• The *start configuration* is $q_0w$
  where $w$ is the input.
3.1 Turing Machines

- The start configuration is $q_0w$ where $w$ is the input.

- A configuration is *accepting configuration* if its state is the accepting state.
3.1 Turing Machines

- The *start configuration* is $q_0w$ where $w$ is the input.
- A configuration is *accepting configuration* if its state is the accepting state.
- A configuration is *rejecting configuration* if its state is the rejecting state.
3.1 Turing Machines

- The start configuration is $q_0w$ where $w$ is the input.

- A configuration is accepting configuration if its state is the accepting state.

- A configuration is rejecting configuration if its state is the rejecting state.

- Both accepting and rejecting configurations are halting configurations.
3.1 Turing Machines

Definition. A Turing machine \( M \) accepts an input \( w \) if a sequence of configurations \( C_1, C_2, \ldots, C_k \), for some \( k \geq 1 \), exists, such that

1. \( C_1 \) is the start configuration of \( M \) on input \( w \),
2. \( C_i \) yields \( C_{i+1} \), for \( i = 1, 2, \ldots, k-1 \), and
3. \( C_k \) is an accepting configuration.

The collection of strings accepted by \( M \) is called

• the language of \( M \), or
• the language recognized by \( M \).

It is denoted with \( L(M) \).

Note: for \( x \in L(M) \), \( M \) always halts; but for \( x \notin L(M) \), \( M \) may or may not halt.
3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k-1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- the language of $M$, or
- the language recognized by $M$.

It is denoted with $L(M)$. Note: for $x \in L(M)$, $M$ always halts; but for $x \not\in L(M)$, $M$ may or may not halt.
3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$, 

...
3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
Definition. A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- the language of $M$, or
- the language recognized by $M$.

It is denoted with $L(M)$. Note: for $x \in L(M)$, $M$ always halts; but for $x \not\in L(M)$, $M$ may or may not halt.
3.1 Turing Machines

Definition. A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- the language of $M$, or

...
3.1 Turing Machines

Definition. A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- the language of $M$, or
- the language recognized by $M$.  

Note: for $x \in L(M)$, $M$ always halts; but for $x \not\in L(M)$, $M$ may or may not halt.
3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- *the language of* $M$, or
- *the language recognized by* $M$. 
3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- *the language of $M*$, or
- *the language recognized by $M*$.

It is denoted with $L(M)$. 

3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- *the language of $M$, or*
- *the language recognized by $M$.*

It is denoted with $L(M)$.

**Note:**
for $x \in L(M)$, $M$ always halts;
3.1 Turing Machines

**Definition.** A Turing machine $M$ accepts an input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$, for some $k \geq 1$ exists, such that

1. $C_1$ is the start configuration of $M$ on input $w$,
2. $C_i$ yields $C_{i+1}$, for $i = 1, 2, \ldots, k - 1$, and
3. $C_k$ is an accepting configuration.

The collection of strings accepted by $M$ is called

- the language of $M$, or
- the language recognized by $M$.

It is denoted with $L(M)$.

**Note:**
for $x \in L(M)$, $M$ always halts;
but for $x \notin L(M)$, $M$ may or may not halt.
3.1 Turing Machines

Definition 3.5
A language \( A \) is Turing recognizable if \( A \) can be recognized by some Turing machine.

A Turing machine decides if it halts. It is called a decider.

Definition 3.6
A language \( A \) is Turing decidable or simply decidable if some Turing machine decides \( A \).

Other terms used:
• Turing recognizable = recursively enumerable
• Turing decidable = recursive
3.1 Turing Machines

Two classes of languages:

Definition 3.5
A language \( A \) is Turing recognizable if \( A \) can be recognized by some Turing machine.

A Turing machine decides if it halts. It is called a decider.

Definition 3.6
A language \( A \) is Turing decidable or simply decidable if some Turing machine decides \( A \).

Other terms used:
- Turing recognizable = recursively enumerable
- Turing decidable = recursive
3.1 Turing Machines

Two classes of languages:

**Definition 3.5** A language $A$ is **Turing recognizable** if $A$ can be recognized by some Turing machine.
3.1 Turing Machines

Two classes of languages:

**Definition 3.5** A language $A$ is **Turing recognizable** if $A$ can be recognized by some Turing machine.

A Turing machine *decides* if it halts. It is called a *decider*. 
3.1 Turing Machines

Two classes of languages:

**Definition 3.5** A language $A$ is **Turing recognizable** if $A$ can be recognized by some Turing machine.

A Turing machine *decides* if it halts. It is called a *decider*.

**Definition 3.6** A language $A$ is **Turing decidable** or simply *decidable* if some Turing machine decides $A$.
3.1 Turing Machines

Two classes of languages:

**Definition 3.5** A language $A$ is **Turing recognizable** if $A$ can be recognized by some Turing machine.

A Turing machine *decides* if it halts. It is called a *decider*.

**Definition 3.6** A language $A$ is **Turing decidable** or simply *decidable* if some Turing machine decides $A$.

**Other terms used:**

- Turing recognizable = *recursively enumerable*
3.1 Turing Machines

Two classes of languages:

**Definition 3.5** A language $A$ is *Turing recognizable* if $A$ can be recognized by some Turing machine.

A Turing machine *decides* if it halts. It is called a *decider*.

**Definition 3.6** A language $A$ is *Turing decidable* or simply *decidable* if some Turing machine decides $A$.

Other terms used:

- Turing recognizable = *recursively enumerable*
- Turing decidable = *recursive*
3.1 Turing Machines

Examples of Turing machines

Example 3.7, recognize language \( \{0^n | n \geq 0\} \) consists of strings: 0, 00, 0000, 00000000.

Idea: \( 2^n \) can be repeatedly divided by 2.

- Accept if the number is 1,
- Reject otherwise.
Examples of Turing machines
Example 3.7, recognize language \( \{0^{2^n} \mid n \geq 0\} \)
Examples of Turing machines
Example 3.7, recognize language \( \{0^{2^n} \mid n \geq 0\} \)

consists of strings: 0, 00, 0000, 00000000.
Examples of Turing machines

Example 3.7, recognize language \( \{0^{2^n} \mid n \geq 0\} \)

- consists of strings: 0, 00, 0000, 00000000.
- idea: \( 2^n \) can be repeatedly divided by 2. If at any iteration of division, the number of 0s is odd...
Examples of Turing machines
Example 3.7, recognize language \( \{0^{2^n} \mid n \geq 0\} \)

consists of strings: 0, 00, 0000, 00000000.

idea: \(2^n\) can be repeatedly divided by 2.
if at any iteration of division, the number of 0s is odd
• accept if the number is 1,
3.1 Turing Machines

Examples of Turing machines
Example 3.7, recognize language \( \{0^{2^n} \mid n \geq 0\} \)

consists of strings: 0, 00, 0000, 00000000.

idea: \( 2^n \) can be repeatedly divided by 2.
   if at any iteration of division, the number of 0s is odd
   • accept if the number is 1,
   • reject otherwise.
Examples of Turing machines
Example 3.7, recognize language \( \{0^{2^n} \mid n \geq 0\} \)

consists of strings: 0, 00, 0000, 00000000.

idea: \( 2^n \) can be repeatedly divided by 2.
   if at any iteration of division, the number of 0s is odd
   • accept if the number is 1,
   • reject otherwise.
More detailed steps for recognition of \{0, 2^n | n ≥ 0\}:

1. Scan left to right the tape, cross off every other 0 symbol,
2. If in step 1, the tape contains a single 0, accept.
3. If in step 1, the tape contains \(k\) 0s, for odd \(k ≥ 3\), reject.
4. Return the head to the leftmost position,
5. Goto step 1.
More detailed steps for recognition of \( \{0^{2^n} \mid n \geq 0\} \)
3.1 Turing Machines

More detailed steps for recognition of \( \{0^{2^n} \mid n \geq 0\} \)

On the input string \( w \):
3.1 Turing Machines

More detailed steps for recognition of \( \{0^{2^n} \mid n \geq 0\} \)

On the input string \( w \):

1. Scan left to right the tape, cross off every other 0 symbol,
More detailed steps for recognition of \( \{0^{2^n} | n \geq 0\} \)

On the input string \( w \):

1. Scan left to right the tape, cross off every other 0 symbol,
2. If in step 1, the tape contains a single 0, accept.
3.1 Turing Machines

More detailed steps for recognition of \( \{0^{2^n} \mid n \geq 0\} \)

On the input string \( w \):

1. Scan left to right the tape, cross off every other 0 symbol,
2. If in step 1, the tape contains a single 0, accept.
3. If in step 1, the tape contains \( k \) 0s, for odd \( k \geq 3 \), reject.
More detailed steps for recognition of \( \{0^{2^n} \mid n \geq 0\} \)

On the input string \( w \):

1. Scan left to right the tape, cross off every other 0 symbol,
2. If in step 1, the tape contains a single 0, accept.
3. If in step 1, the tape contains \( k \) 0s, for odd \( k \geq 3 \), reject.
4. Return the head to the leftmost position,
More detailed steps for recognition of \( \{0^{2^n} \mid n \geq 0\} \)

On the input string \( w \):

1. Scan left to right the tape, cross off every other 0 symbol,
2. If in step 1, the tape contains a single 0, accept.
3. If in step 1, the tape contains \( k \) 0s, for odd \( k \geq 3 \), reject.
4. Return the head to the leftmost position,
5. Goto step 1.
3.1 Turing Machines
3.1 Turing Machines

Implementation-level description:
3.1 Turing Machines

Implementation-level description:

\[ M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject}) \]

\[ Q = \{ q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject} \}, \]
3.1 Turing Machines

Implementation-level description:

$$M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$$

$$Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\},$$

$$\Sigma = \{0\},$$
3.1 Turing Machines

Implementation-level description:

\[ M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}}) \]

- \( Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\} \),
- \( \Sigma = \{0\} \),
- \( \Gamma = \{0, \times, \sqcup\} \).
3.1 Turing Machines

Implementation-level description:

\[ M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}}) \]

\[ Q = \{ q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}} \}, \]
\[ \Sigma = \{0\}, \]
\[ \Gamma = \{0, \times, \square\}, \]

Start state is \( q_1 \),
3.1 Turing Machines

Implementation-level description:

\[ M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}}) \]

\[ Q = \{ q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}} \}, \]

\[ \Sigma = \{ 0 \}, \]

\[ \Gamma = \{ 0, \times, \sqcup \}, \]

Start state is \( q_1 \),

Accept state is \( q_{\text{accept}} \); reject state is \( q_{\text{reject}} \), and
3.1 Turing Machines

Implementation-level description:

\[ M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}}) \]

\[ Q = \{ q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}} \}, \]
\[ \Sigma = \{ 0 \}, \]
\[ \Gamma = \{ 0, \times, \sqcup \}, \]

Start state is \( q_1 \),

Accept state is \( q_{\text{accept}} \), reject state is \( q_{\text{reject}} \), and

\( \delta \) is described with a state diagram (Figure 3.8, page 172).
3.1 Turing Machines

- from \( q_1 \) mark the first 0 as \( \sqcup \) to indicate the leftmost, go to \( q_2 \)
- if no first 0, go to \( q_\text{reject} \)
- at \( q_2 \) move right, skips all \( \times \)'s, if no more 0, go to \( q_\text{accept} \), or
- cross the first encountered 0, go to \( q_3 \)
- \( q_3 \) together with \( q_4 \) skip all \( \times \)'s, cross every other 0
- if not even number of 0, from \( q_4 \) go to \( q_\text{reject} \)
- at \( q_3 \) reach the rightmost, move left, go to \( q_5 \)
- move left, \( q_5 \) skips all \( \times \)'s and 0's
- reach the leftmost, go to \( q_2 \), then the whole process repeats from
3.1 Turing Machines

- from $q_1$ mark the first 0 as ⊔ to indicate the leftmost, go to $q_2$
- if no first 0, go to $q_{reject}$
- at $q_2$ move right, skips all ×'s, if no more 0, go to $q_{accept}$, or
- - cross the first encountered 0, go to $q_3$
- - $q_3$ together with $q_4$ skip all ×'s, cross every other 0
- - if not even number of 0, from $q_4$ go to $q_{reject}$
- - at $q_3$ reach the rightmost, move left, go to $q_5$
- - move left, $q_5$ skips all ×'s and 0's
- - reach the leftmost, go to $q_2$, then the whole process repeats from •
- from $q_1$ mark the first 0 as $\sqcup$ to indicate the leftmost, go to $q_2$
- if no first 0, go to $q_{reject}$
  • at $q_2$ move right, skips all $\times$’s, if no more 0, go to $q_{accept}$, or
  - cross the first encountered 0, go to $q_3$
- $q_3$ together with $q_4$ skip all $\times$’s, cross every other 0
- if not even number of 0, from $q_4$ go to $q_{reject}$
- at $q_3$ reach the rightmost, move left, go to $q_5$
- move left, $q_5$ skips all $\times$’s and 0’s
- reach the leftmost, go to $q_2$, then the whole process repeats from $\bullet$
3.1 Turing Machines

Trace the execution of the TM on input 0000 using configuration changes.

Note: the input is accepted if only one 0 left (representing value 1). So one that 0 is first replaced with ⊓ at q₁.
3.1 Turing Machines
Trace the execution of the TM on input 0000 using configuration changes.

\[
\begin{array}{ccc}
q_1 & 0000 & q_5 \xrightarrow{x} 0 x \\
\uparrow q_2 & 000 & q_5 \xrightarrow{x} 0 x \\
\uparrow x & q_3 & 00 \xrightarrow{q_2} 0 x \\
\uparrow x & 0 q_4 & 0 \xrightarrow{q_5} 0 x \\
\uparrow x & 0 x q_3 & \xrightarrow{q_5} x x \\
\uparrow x & 0 x q_5 & \xrightarrow{q_5} x x \\
\uparrow x & q_5 & 0 \xrightarrow{q_5} x x \\
\end{array}
\]
3.1 Turing Machines

Trace the execution of the TM on input 0000 using configuration changes.

Note: the input is accepted if only one 0 left (representing value 1). So one that 0 is first replaced with ⊥ at $q_1$. 

![Turing Machine Diagram]
3.1 Turing Machines

Example 3.9 TM \( M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}}) \) to recognize language \( \{ w \# w | w \in \{0, 1\}^* \} \).

Explanation on the next page
3.1 Turing Machines

Example 3.9 TM $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ to recognize language $\{w\#w \mid w \in \{0, 1\}^*\}$.

Explanation on the next page
3.1 Turing Machines

- from $q_1$, go to $q_2$ if read 0; go to $q_3$ if read 1, cross current symbol • from $q_2$, move right, skip all 0's and 1's, until reach #, go to $q_4$,
- from $q_4$, move right, skip all ×'s until 0, go to $q_6$, move left
- from $q_6$, move left, skip all 0's, 1's, ×'s, until #, go to $q_7$, move left
- from $q_7$, move left, skip all 0's, 1's until ×, go to $q_1$, move right
• from $q_3$, symmetrically similar to from $q_2$.
- from $q_1$, if # (no more 0's or 1's on its left), go to $q_8$, move right
- from $q_8$, if no more 0's or 1's, go to $q$ accept
3.1 Turing Machines

- from $q_1$, go to $q_2$ if read 0; go to $q_3$ if read 1, cross current symbol
- from $q_2$, move right, skip all 0’s and 1’s, until reach #, go to $q_4$,
- from $q_4$, move right, skip all $\times$’s until 0, go to $q_6$, move left
- from $q_6$, move left, skip all 0’s, 1’s, $\times$’s, until #, go to $q_7$, move left
- from $q_7$, move left, skip all 0’s, 1’s until $\times$, go to $q_1$, move right
- from $q_3$, symmetrically similar to from $q_2$.
- from $q_1$, if # (no more 0’s or 1’s on its left), go to $q_8$, move right
- from $q_8$, if no more 0’s or 1’s, go to $q_{accept}$
3.1 Turing Machines

Turing machines can accomplish complicated tasks:

- **e.g.,** addition of two integers
  - **1**-x (unary) represents input
  - **1**x + 1**y** = 1**z**, accept iff **x** + **y** = **z**.

- **e.g.,** 111 + 11 = 11111, how would a TM do it?

- **binary representation,** **x**, **y**, **z** ∈ {0, 1}:
  - input **x** + **y** = **z**, accept iff **x** + **y** = **z**.

- **e.g.,** 011 + 10 = 101, how would a TM do it?

Turing machines can have output, i.e., compute functions, not just predicates.
Turing machines can accomplish complicated tasks:

- Represent integers, e.g., $1_x$ (unary) represents $x$.
- Addition: $x + y = z$, accept iff $x + y = z$.
  - Example: $111 + 11 = 11111$.

- Binary representation: $x, y, z \in \{0, 1\}^*$.
- Addition: $x + y = z$, accept iff $x + y = z$.
  - Example: $011 + 10 = 101$. How would a TM do it?

Turing machines can have output, i.e., compute functions, not just predicates.
3.1 Turing Machines

Turing machines can accomplish complicated tasks:

- e.g., addition of two integers

Turing machines can have output, i.e., compute functions, not just predicates.
3.1 Turing Machines

Turing machines can accomplish complicated tasks:

- e.g., addition of two integers
  
  - represent integers, $1^x$ (unary) represents $x$
  
  input $1^x + 1^y = 1^z$, accept iff $x + y = z$.
  
  e.g., $111+11=1111$, how would a TM do it?
3.1 Turing Machines

Turing machines can accomplish complicated tasks:

e.g., addition of two integers

• represent integers, $1^x$ (unary) represents $x$
  input $1^x + 1^y = 1^z$, accept iff $x + y = z$.
  e.g., $111 + 11 = 11111$, how would a TM do it?

• binary representation, $x, y, z \in \{0, 1\}^*$
  input $x + y = z$, accept iff $x + y = z$.
  e.g., $011 + 10 = 101$, how would a TM do it?
3.1 Turing Machines

Turing machines can accomplish complicated tasks:

- e.g., addition of two integers
  - represent integers, $1^x$ (unary) represents $x$
    input $1^x + 1^y = 1^z$, accept iff $x + y = z$.
    e.g., $111 + 11 = 11111$, how would a TM do it?
  - binary representation, $x, y, z \in \{0, 1\}^*$
    input $x + y = z$, accept iff $x + y = z$.
    e.g., $011 + 10 = 101$, how would a TM do it?

Turing machines can have output,
  i.e., compute functions, not just predicates.
3.2 Variants of Turing Machines

- Read head may not move after reading a symbol: \{L, R, S\}
- Multiple tapes
- Two-way infinite input tape
- Random access input tape (with an address tape)
- With output tape
3.2 Variants of Turing Machines

- read head may not move after reading a symbol;
- multiple tapes;
- two-way infinite input tape;
- random access input tape (with an address tape);
- with output tape;
3.2 Variants of Turing Machines

- read head may not move after reading a symbol; \( \{L, R, S\} \)
3.2 Variants of Turing Machines

- read head may not move after reading a symbol; \( \{L, R, S\} \)
- multiple tapes;
3.2 Variants of Turing Machines

- read head may not move after reading a symbol; \( \{L, R, S\} \)
- multiple tapes;
- two-way infinite input tape;
3.2 Variants of Turing Machines

- read head may not move after reading a symbol; \( \{L, R, S\} \)
- multiple tapes;
- two-way infinite input tape;
- random access input tape (with an address tape)
3.2 Variants of Turing Machines

- read head may not move after reading a symbol; \( \{L, R, S\} \)
- multiple tapes;
- two-way infinite input tape;
- random access input tape (with an address tape)
- with output tape;
3.2 Variants of Turing Machines

Multitape Turing machines

- Have $k$ tapes, each with a head to read and write
- $k - 1$ tapes start out blank
- Transition function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R,S\}^k$

Theorem 3.13
Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof idea:
- assigning space for $k$ tapes using delimiter, e.g., #
- shift right to get more space
- how about read head positions?
3.2 Variants of Turing Machines

Multitape Turing machines

Theorem 3.13
Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof idea:
- assigning space for $k$ tapes using delimiter, e.g., #
- shift right to get more space
- how about read head positions?
3.2 Variants of Turing Machines

Multitape Turing machines

- Have \( k \) tapes, each with a head to read and write
- \( k - 1 \) tapes start out blank
- Transition function \( \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k \)
3.2 Variants of Turing Machines

Multitape Turing machines

- Have $k$ tapes, each with a head to read and write
- $k - 1$ tapes start out blank
- Transition function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

**Theorem 3.13** Every multi-tape Turing machine has an equivalent single-tape Turing machine.
3.2 Variants of Turing Machines

Multitape Turing machines

- Have $k$ tapes, each with a head to read and write
- $k - 1$ tapes start out blank
- Transition function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

Theorem 3.13 Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof idea:
- assigning space for $k$ tapes using delimiter, e.g., #
- shift right to get more space
- how about read head positions?
3.2 Variants of Turing Machines

Nondeterministic Turing machines

• $\delta$: $Q \times \Gamma \rightarrow P(2^{Q \times \Gamma \times \{L,R\}})$

  e.g., design a TM to recognize language $L = \{ww : w \in \{0,1\}^*\}$

  we have seen a deterministic TM for $\{w#w : w \in \{0,1\}^*\}$:
3.2 Variants of Turing Machines

Nondeterministic Turing machines

- \( \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}) \)
3.2 Variants of Turing Machines

Nondeterministic Turing machines

- \( \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}) \)

  e.g., design a TM to recognize language \( L = \{ww : w \in \{0, 1\}^*\} \)
3.2 Variants of Turing Machines

Nondeterministic Turing machines

- $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$

e.g., design a TM to recognize language $L = \{ww : w \in \{0, 1\}^*\}$

we have seen a deterministic TM for $\{w\#w : w \in \{0, 1\}^*\}$:

![Diagram of a nondeterministic Turing machine](image-url)
3.2 Variants of Turing Machines

To recognize language \( L = \{ww : w \in \{0, 1\}^*\} \), we may nondeterministically guess the mid position. How may we modify the TM for our need?
3.2 Variants of Turing Machines

To recognize language $L = \{ww : w \in \{0, 1\}^*\}$, we may nondeterministically guess the mid position.
3.2 Variants of Turing Machines

To recognize language $L = \{ww : w \in \{0, 1\}^*\}$, we may nondeterministically guess the mid position.

How may we modify the TM for our need?
3.2 Variants of Turing Machines

The following portion of a TM transforms input $x_1 x_2 \ldots x_n$ to $x_1 x_2 \ldots x_{k-1} \# x_k \ldots x_n$ where $k$ is chosen nondeterministically.
3.2 Variants of Turing Machines

The following portion of a TM transforms input

\[ x_1 x_2 \ldots x_n \] to

\[ x_1 x_2 \ldots x_{k-1} \# x_k \ldots x_n \] where \( k \) is chosen nondeterministically.
3.2 Variants of Turing Machines

The following portion of a TM transforms input

\[ x_1 x_2 \ldots x_n \]
3.2 Variants of Turing Machines

The following portion of a TM transforms input

\[ x_1 x_2 \ldots x_n \]

to

\[ x_1 x_2 \ldots x_{k-1} \# x_k \ldots x_n \]

where \( k \) is chosen nondeterministically.
3.2 Variants of Turing Machines

The following portion of a TM transforms input

\[ x_1 x_2 \ldots x_n \]

to

\[ x_1 x_2 \ldots x_{k-1} \# x_k \ldots x_n \]

where \( k \) is chosen nondeterministically.
3.2 Variants of Turing Machines

• $P_1$ marks the leftmost position by replacing 0 with $Z$ and 1 with $W$;
• $P_2$ moves read-head to the rightmost;
• $P_3, P_4, P_5, P_6$ shift symbols one position down to the right, iteratively;
• at the end of each iteration, $P_6$ nondeterministically decides to keep shifting
  OR
• inserting # and moves to the leftmost (and recovering the leftmost symbol)
3.2 Variants of Turing Machines

- $P_1$ marks the leftmost position by replacing 0 with $Z$ and 1 with $W$;

- $P_2$ moves read-head to the rightmost;

- $P_3$, $P_4$, $P_5$, $P_6$ shift symbols one position down to the right, iteratively;

- at the end of each iteration, $P_6$ nondeterministically decides to keep shifting OR inserting $#$ and moves to the leftmost (and recovering the leftmost symbol).
3.2 Variants of Turing Machines

- $P_1$ marks the leftmost position by replacing 0 with $Z$ and 1 with $W$;
- $P_2$ moves read-head to the rightmost;
- $P_3, P_4, P_5, P_6$ shift symbols one position down to the right, iteratively;
- at the end of each iteration, $P_6$ nondeterministically decides to keep shifting OR inserting $#$ and moves to the leftmost (and recovering the leftmost symbol).
3.2 Variants of Turing Machines

- $P_1$ marks the leftmost position by replacing 0 with $Z$ and 1 with $W$;
- $P_2$ moves read-head to the rightmost;
- $P_3$, $P_4$, $P_5$, $P_6$ shifts symbols one position down to the right, iteratively;
- $P_7$ nondeterministically decides to keep shifting or inserting $\#$ and moves to the leftmost (and recovering the leftmost symbol).
3.2 Variants of Turing Machines

- $P_1$ marks the leftmost position by replacing 0 with $Z$ and 1 with $W$;
- $P_2$ moves read-head to the rightmost;
- $P_3$, $P_4$, $P_5$, $P_6$ shifts symbols one position down to the right, iteratively;
- at the end of each iteration, $P_6$ nondeterministically decides to keep shifting OR
3.2 Variants of Turing Machines

- $P_1$ marks the leftmost position by replacing 0 with $Z$ and 1 with $W$;
- $P_2$ moves read-head to the rightmost;
- $P_3$, $P_4$, $P_5$, $P_6$ shifts symbols one position down to the right, iteratively;
- at the end of each iteration, $P_6$ nondeterministically decides to keep shifting OR
- inserting $\#$ and moves to the leftmost (and recovering the leftmost symbol)
3.2 Variants of Turing Machines

Another example of nondeterministic TM.

To recognize language $L = \{u \# w : u \text{ is a substring of } w \}$.

Construct a TM $M$ for $L$ as follows

1. on input $u \# w$,
2. $M$ nondeterministically chooses a position $k$ in $w$, and
3. compare $u$ with the substring beginning at position $k$;
4. $M$ accepts $u \# w$ iff $u$ and the substring match.
Another example of nondeterministic TM.

Construct a TM $M$ for $L$ as follows

- on input $u \# w$,
- $M$ nondeterministically chooses a position $k$ in $w$, and
- compare $u$ with the substring beginning at position $k$;
- $M$ accepts $u \# w$ iff $u$ and the substring match.
Another example of nondeterministic TM.

To recognize language $L = \{u\#w : u \text{ is a substring of } w\}$.
3.2 Variants of Turing Machines

Another example of nondeterministic TM.

To recognize language \( L = \{ u \# w : u \) is a substring of \( w \} \).

Construct a TM \( M \) for \( L \) as follows

• on input \( u \# w \),
• \( M \) nondeterministically chooses a position \( k \) in \( w \), and
• compare \( u \) with the substring beginning at position \( k \);
• \( M \) accepts \( u \# w \) iff \( u \) and the substring match.
3.2 Variants of Turing Machines

Another example of nondeterministic TM.
To recognize language $L = \{u\#w : u \text{ is a substring of } w\}$.
Construct a TM $M$ for $L$ as follows

- on input $u\#w$, 
3.2 Variants of Turing Machines

Another example of nondeterministic TM.

To recognize language $L = \{ u\#w : u$ is a substring of $w \}$. 

Construct a TM $M$ for $L$ as follows:

- on input $u\#w$,
- $M$ nondeterministically chooses a position $k$ in $w$, and compare $u$ with the substring beginning at position $k$;
3.2 Variants of Turing Machines

Another example of nondeterministic TM.

To recognize language $L = \{u\#w : u$ is a substring of $w\}$.

Construct a TM $M$ for $L$ as follows

- on input $u\#w$,
- $M$ nondeterministically chooses a position $k$ in $w$, and compare $u$ with the substring beginning at position $k$;
- $M$ accepts $u\#w$ iff $u$ and the substring match.
3.2 Variants of Turing Machines

Theorem 3.16

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:
- Computation of NTM is a tree (possibly infinite)
- Each node is a configuration, the root is the start configuration
- From a parent, there can be $m$ children configurations
- Breadth-first-search (why depth-first-search may not work?)
3.2 Variants of Turing Machines

**Theorem 3.16** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

*Proof idea:*
- computation of NTM is a tree (possibly infinite)
Theorem 3.16 Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:
- computation of NTM is a tree (possibly infinite)
  (what does a DTM computation look like)
Theorem 3.16 Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:
- computation of NTM is a tree (possibly infinite)
  (what does a DTM computation look like)
- each node is a configuration, the root is the start configuration
Theorem 3.16 Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:
- computation of NTM is a tree (possibly infinite)
  (what does a DTM computation look like)
- each node is a configuration, the root is the start configuration
- from a parent, there can be \( m \) children configurations
Theorem 3.16 Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:
- computation of NTM is a tree (possibly infinite)
  (what does a DTM computation look like)
- each node is a configuration, the root is the start configuration
- from a parent, there can be $m$ children configurations
- breadth-first-search (why depth-first-search may not work?)
3.2 Variants of Turing Machines

Technical details for the proof:

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree. Note that given a configuration, all children configurations are known.

- Use two tapes:
  - On path tape: the path from root to current configuration is stored, e.g., in the form of 1 2 1 3 2 3 at level 6.
  - On simulation tape: given the path, computation is deterministically simulated.
3.2 Variants of Turing Machines

Technical details for the proof:

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree.
3.2 Variants of Turing Machines

Technical details for the proof:

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree. Note that given a configuration, all children configurations are known.
3.2 Variants of Turing Machines

*Technical details for the proof:*

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree.

  *Note that given a configuration, all children configurations are known.*

- Use two tapes:
Technical details for the proof:

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree.

  Note that given a configuration, all children configurations are known.

- Use two tapes:
  
  - on path tape: the path from root to current configuration is stored e.g., in the form of 1 2 1 3 2 3 at level 6.
3.2 Variants of Turing Machines

*Technical details for the proof:*

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree.

  **Note that given a configuration, all children configurations are known.**

- Use two tapes:
  - on path tape: the path from root to current configuration is stored e.g., in the form of 1 2 1 3 2 3 at level 6.
  - on simulation tape: given the path, computation is deterministically simulated.
3.2 Variants of Turing Machines

Technical details for the proof:

- A DTM simulates the computation of a given NTM on input by searching through the computation/configuration tree.

Note that given a configuration, all children configurations are known.

- Use two tapes:
  - on path tape: the path from root to current configuration is stored e.g., in the form of 1 2 1 3 2 3 at level 6.
  - on simulation tape: given the path, computation is deterministically simulated.
3.2 Variants of Turing Machines

Turing Enumerators

A Turing machine that prints strings on the output tape, separated by a special symbol, say #, is called an enumerator.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
- The strings printed out can be in any order.
- The collection can be infinite.

Theorem 3.21

A language is Turing recognizable if and only if there is an Turing enumerator that enumerates it.

There are two implications to prove:

- Turing enumerable $\Rightarrow$ Turing recognizable.
- Turing recognizable $\Rightarrow$ Turing enumerable.
3.2 Variants of Turing Machines

**Turing Enumerators**

A Turing machine that prints strings on the output tape, separated by a special symbol, say #, is called an enumerator.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
- The strings printed out can be in any order.
- The collection can be infinite.

**Theorem 3.21**

A language is Turing recognizable if and only if there is a Turing enumerator that enumerates it.

There are two implications to prove:

1. Turing enumerable $\Rightarrow$ Turing recognizable.
2. Turing recognizable $\Rightarrow$ Turing enumerable.
3.2 Variants of Turing Machines

Turing Enumerators

A Turing machine that prints strings on the output tape, separated by a special symbol, say #, is called an enumerator.

Theorem 3.21

A language is Turing recognizable if and only if there is an Turing enumerator that enumerates it.

There are two implications to prove:

1. Turing enumerable $\Rightarrow$ Turing recognizable.
2. Turing recognizable $\Rightarrow$ Turing enumerable.
3.2 Variants of Turing Machines

**Turing Enumerators**

A Turing machine that prints strings on the *output* tape, separated by a special symbol, say #, is called an *enumerator*.

- It starts from an empty input tape.

**Theorem 3.21**

A language is Turing recognizable if and only if there is a Turing enumerator that enumerates it.

There are two implications to prove:

1. Turing enumerable \( \Rightarrow \) Turing recognizable.
2. Turing recognizable \( \Rightarrow \) Turing enumerable.
Turing Enumerators

A Turing machine that prints strings on the output tape, separated by a special symbol, say $\#$, is called an enumerator.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
3.2 Variants of Turing Machines

Turing Enumerators

A Turing machine that prints strings on the output tape, separated by a special symbol, say #, is called an enumerator.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
- The strings printed out can be in any order.

Theorem 3.21
A language is Turing recognizable if and only if there is an Turing enumerator that enumerates it.

There are two implications to prove:

1. Turing enumerable $\implies$ Turing recognizable.
2. Turing recognizable $\implies$ Turing enumerable.
3.2 Variants of Turing Machines

Turing Enumerators

A Turing machine that prints strings on the output tape, separated by a special symbol, say $\#$, is called an enumerator.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
- The strings printed out can be in any order.
- The collection can be infinite.

Theorem 3.21: A language is Turing recognizable if and only if there is a Turing enumerator that enumerates it.

There are two implications to prove:

(1) Turing enumerable $\equiv$ Turing recognizable.

(2) Turing recognizable $\Rightarrow$ Turing enumerable.
3.2 Variants of Turing Machines

**Turing Enumerators**

A Turing machine that prints strings on the *output* tape, separated by a special symbol, say #, is called an *enumerator*.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
- The strings printed out can be in any order.
- The collection can be infinite.

**Theorem 3.21** A language is Turing recognizable if and only if there is an Turing enumerator that enumerates it.
3.2 Variants of Turing Machines

**Turing Enumerators**

A Turing machine that prints strings on the *output* tape, separated by a special symbol, say #, is called an **enumerator**.

- It starts from an empty input tape.
- The set of all strings printed is the language enumerated by the TM.
- The strings printed out can be in any order.
- The collection can be infinite.

**Theorem 3.21** A language is Turing recognizable if and only if there is an Turing enumerator that enumerates it.

*There are two implications to prove:*
3.2 Variants of Turing Machines

Turing Enumerators

A Turing machine that prints strings on the output tape, separated by a special symbol, say #, is called an enumerator.

• It starts from an empty input tape.
• The set of all strings printed is the language enumerated by the TM.
• The strings printed out can be in any order.
• The collection can be infinite.

Theorem 3.21 A language is Turing recognizable if and only if there is an Turing enumerator that enumerates it.

There are two implications to prove:

(1) Turing enumerable $\implies$ Turing recognizable.
(2) Turing recognizable $\implies$ Turing enumerable.
3.2 Variants of Turing Machines

Proof ideas for:

(1) Turing enumerable $\Rightarrow$ Turing recognizable.

- Assume $E$ a Turing enumerator for language $L$,
- Construct Turing machine $M$ to read the given input $w$ so $M$ follows the work of $E$ that prints strings on the output tape;
- For every string $x$ sequentially printed out, $M$ compare $w$ with $x$;
- If $w$ is in $L(E)$, it will get printed on the tape eventually, so will be accepted by $M$ and $M$ will halt.
- If $w$ is NOT in $L(E)$, it will never get printed on the output tape, so $M$ will not halt - meaning reject $w$. 
3.2 Variants of Turing Machines

Proof ideas for:

(1) Turing enumerable $\implies$ Turing recognizable.
3.2 Variants of Turing Machines

Proof ideas for:

(1) Turing enumerable \(\implies\) Turing recognizable.
   - assume \(E\) a Turing enumerator for language \(L\),
   - if \(w\) is in \(L(E)\), it will get printed on the tape eventually, so will be accepted by \(M\) and \(M\) will halt.
   - if \(w\) is NOT in \(L(E)\), it will never get printed on the output tape, so \(M\) will not halt - meaning reject \(w\).
Proof ideas for:

(1) Turing enumerable $\implies$ Turing recognizable.

- assume $E$ a Turing enumerator for language $L$,
- construct Turing machine $M$ to read the given input $w$ so
  - $M$ follows the work of $E$ that prints strings on the output tape;
  - for every string $x$ sequentially printed out, $M$ compare $w$ with $x$
3.2 Variants of Turing Machines

Proof ideas for:

(1) Turing enumerable $\implies$ Turing recognizable.

- assume $E$ a Turing enumerator for language $L$,
- construct Turing machine $M$ to read the given input $w$ so
  - $M$ follows the work of $E$ that prints strings on the output tape;
  - for every string $x$ sequentially printed out, $M$ compare $w$ with $x$

- if $w$ is in $L(E)$, it will get printed on the tape eventually,
  so will be accepted by $M$ and $M$ will halt.

- if $w$ is NOT in $L(E)$, it will never get printed on the output tape,
  so $M$ will not halt - meaning reject $w$. 
3.2 Variants of Turing Machines

Proof ideas for:

(2) Turing recognizable ⇔ Turing enumerable.

• assume \( M \) a Turing machine that recognizes language \( L \).
• construct enumerator \( E \) to enumerate \( L(M) \):
  - for each string \( w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, ... \), \( E \) follows the work of \( M \) on \( w \), if \( M \) accepts \( w \), \( E \) prints \( w \) on the output tape.
  

but would this work? NO! because on some strings, \( M \) may never halt!

consider \( 00 \notin L \) but \( 11 \in L \)
how to fix?

assume \( s_1 = \epsilon, s_2 = 0, s_3 = 1, s_4 = 00, ... \) for \( i = 1, 2, ... \), \( E \) follows \( M_i \) steps on each of \( s_1, s_2, ..., s_i \) if \( M \) accepts \( s_j \), \( E \) prints \( s_j \).

This ensures that for any \( x \in L \), \( x \) gets tested on \( M \), and thus gets printed by \( E \).
3.2 Variants of Turing Machines

3.3 Definition of Algorithms *Proof ideas for:

(2) Turing recognizable $\implies$ Turing enumerable.
3.2 Variants of Turing Machines

3.3 Definition of Algorithms *Proof ideas for:*

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$. 

...
3.3 Definition of Algorithms *Proof ideas for:*

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
3.2 Variants of Turing Machines

3.3 Definition of Algorithms *Proof ideas for*:

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$, but would this work?
  
  NO! because on some strings, $M$ may never halt!
  
  consider $00 \not\in L$ but $11 \in L$
  
  how to fix?

  assume $s_1 = \epsilon$, $s_2 = 0$, $s_3 = 1$, $s_4 = 00$, ...
  
  for $i = 1, 2, 3, \ldots$, $E$ follows $M_i$ steps on each of $s_1, s_2, \ldots, s_i$ if $M$ accepts $s_j$, $E$ prints $s_j$.
  
  This ensures that for any $x \in L$, $x$ gets tested on $M$, and thus gets printed by $E$. 
3.2 Variants of Turing Machines

3.3 Definition of Algorithms \textit{Proof ideas for:}

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    $E$ follows the work of $M$ on $w$,
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.
3.3 Definition of Algorithms *Proof ideas for:*

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    $E$ follows the work of $M$ on $w$,
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

**but would this work?**
3.2 Variants of Turing Machines

3.3 Definition of Algorithms Proof ideas for:

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    $E$ follows the work of $M$ on $w$,
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

NO!
3.2 Variants of Turing Machines

3.3 Definition of Algorithms  

Proof ideas for:

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,  
    $E$ follows the work of $M$ on $w$,  
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

NO! because on some strings, $M$ may never halt!
3.2 Variants of Turing Machines

3.3 Definition of Algorithms *Proof ideas for:*

(2) Turing recognizable $\iff$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,  
    $E$ follows the work of $M$ on $w$,  
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

  **but would this work?**

  NO! because on some strings, $M$ may never halt!
  - consider $00 \notin L$ but $11 \in L$
3.3 Definition of Algorithms *Proof ideas for:*

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    - $E$ follows the work of $M$ on $w$,
    - if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

**NO!** because on some strings, $M$ may never halt!

consider $00 \notin L$ but $11 \in L$

how to fix?
3.2 Variants of Turing Machines

3.3 Definition of Algorithms \textit{Proof ideas for:}

(2) Turing recognizable $\iff$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$, $E$ follows the work of $M$ on $w$,
  - if $M$ accepts $w$, $E$ prints $w$ on the output tape.

\textbf{but would this work?}

\textbf{NO!} because on some strings, $M$ may never halt!

\textbf{consider} $00 \not\in L$ but $11 \in L$

\textbf{how to fix?} assume $s_1 = \epsilon, s_2 = 0, s_3 = 1, s_4 = 00, \ldots$
3.2 Variants of Turing Machines

3.3 Definition of Algorithms Proof ideas for:

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    $E$ follows the work of $M$ on $w$,
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

NO! because on some strings, $M$ may never halt!

consider $00 \notin L$ but $11 \in L$

how to fix? assume $s_1 = \epsilon, s_2 = 0, s_3 = 1, s_4 = 00, \ldots$
for $i = 1, 2, \ldots$,
3.2 Variants of Turing Machines

3.3 Definition of Algorithms *Proof ideas for:*

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    - $E$ follows the work of $M$ on $w$,
    - if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

NO! because on some strings, $M$ may never halt!

consider $00 \notin L$ but $11 \in L$

how to fix? assume $s_1 = \epsilon, s_2 = 0, s_3 = 1, s_4 = 00, \ldots$
for $i = 1, 2, \ldots$,

$E$ follows $M$ $i$ steps on each of $s_1, s_2, \ldots, s_i$
3.2 Variants of Turing Machines

3.3 Definition of Algorithms Proof ideas for:

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    $E$ follows the work of $M$ on $w$,
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

NO! because on some strings, $M$ may never halt!

consider $00 \notin L$ but $11 \in L$

how to fix? assume $s_1 = \epsilon, s_2 = 0, s_3 = 1, s_4 = 00, \ldots$

for $i = 1, 2, \ldots$,

$E$ follows $M$ $i$ steps on each of $s_1, s_2, \ldots, s_i$

if $M$ accepts $s_j$, $E$ prints $s_j$. 
3.2 Variants of Turing Machines

3.3 Definition of Algorithms \textit{Proof ideas for:}

(2) Turing recognizable $\implies$ Turing enumerable.

- assume $M$ a Turing machine that recognizes language $L$.
- construct enumerator $E$ to enumerate $L(M)$:
  - for each string $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$,
    $E$ follows the work of $M$ on $w$,
    if $M$ accepts $w$, $E$ prints $w$ on the output tape.

but would this work?

NO! because on some strings, $M$ may never halt!

consider $00 \notin L$ but $11 \in L$

how to fix? assume $s_1 = \epsilon, s_2 = 0, s_3 = 1, s_4 = 00, \ldots$

for $i = 1, 2, \ldots$,

$E$ follows $M$ $i$ steps on each of $s_1, s_2, \ldots, s_i$

if $M$ accepts $s_j$, $E$ prints $s_j$.

This ensures that for any $x \in L$, $x$ gets tested on $M$, and
thus gets printed by $E$
3.3 Definition of Algorithms

Recalled there are two kinds of Turing machines:
1. TMs that always halt on all inputs; the instruction set built in such a machine is called an algorithm.
2. Those may not halt on some inputs; Church-Turing Thesis: Intuitive (human) notation of algorithms equals Turing machine notation of algorithms.
3.3 Definition of Algorithms

3.3 Definition of Algorithm

Recalled there are two kinds of Turing machines:

1. Those that always halt on all inputs; the instruction set built in such a machine is called an **algorithm**.

2. Those that may not halt on some inputs; Church-Turing Thesis: Intuitive (human) notation of algorithms equals Turing machine notation of algorithms.
3.3 Definition of Algorithms

3.3 Definition of Algorithm

Recalled there are two kinds of Turing machines: (1) TMs that always halt on all inputs; the instruction set built in such a machine is called an algorithm. (2) Those may not halt on some inputs; Church-Turing Thesis: Intuitive (human) notation of algorithms equals Turing machine notation of algorithms.
3.3 Definition of Algorithms

3.3 Definition of Algorithm

Recalled there are two kinds of Turing machines

(1) TMs that always halt on all inputs;
3.3 Definition of Algorithms

Recalled there are two kinds of Turing machines

(1) TMs that always halt on all inputs;
   - the instruction set built in such a machine is called an algorithm.
3.3 Definition of Algorithms

3.3 Definition of Algorithm

Recalled there are two kinds of Turing machines

(1) TMs that always halt on all inputs;
   • the instruction set built in such a machine is called an algorithm.

(2) those may not halt on some inputs;
3.3 Definition of Algorithms

Recalled there are two kinds of Turing machines

(1) TMs that always halt on all inputs;
   • the instruction set built in such a machine
     is called an algorithm.

(2) those may not halt on some inputs;

Church-Turing Thesis:

Intuitive (human) notation of algorithms

equals

Turing machine notation of algorithms.
3.3 Definition of Algorithms

Recalled there are two kinds of Turing machines

(1) TMs that always halt on all inputs;
   - the instruction set built in such a machine is called an algorithm.

(2) those may not halt on some inputs;

Church-Turing Thesis:
   Intuitive (human) notation of algorithms equals
   Turing machine notation of algorithms.
3.3 Definition of Algorithms

More on Church-Turing Thesis:
Three formal systems are equivalent: logic and recursive functions [Gödel 1933], $\lambda$-calculus [Church 1936] and Turing machine algorithms [Turing 1936].

Three programming (language) paradigms:
- logic programming languages: prolog
- functional programming languages: LISP
- procedural programming languages: C, Java, etc.
3.3 Definition of Algorithms

More on Church-Turing Thesis:
3.3 Definition of Algorithms

More on Church-Turing Thesis:

Three formal systems are equivalent:
- logic and recursive functions [Gödel 1933]
- λ-calculus [Church 1936] and
- Turing machine algorithms [Turing 1936]
3.3 Definition of Algorithms

More on Church-Turing Thesis:

Three formal systems are equivalent:
- logic and recursive functions [Gödel 1933]
- $\lambda$-calculus [Church 1936] and
- Turing machine algorithms [Turing 1936]

Three programming (language) paradigms:
- logic programming languages: prolog
- functional programming languages: LISP
- procedural programming languages: C, Java, etc
3.3 Definition of Algorithms
3.3 Definition of Algorithms

- What problems cannot be solved by algorithms?
3.3 Definition of Algorithms

• What problems cannot be solved by algorithms?
  What problems may not be formulated into
  Turing decidable (i.e., recursive) languages?
3.3 Definition of Algorithms

- **What problems cannot be solved by algorithms?**
  
  What problems may not be formulated into Turing decidable (i.e., recursive) languages?

- **What problems cannot even be solved by Turing machines that may not halt on all inputs?**
3.3 Definition of Algorithms

- **What problems cannot be solved by algorithms?**
  What problems may not be formulated into Turing decidable (i.e., recursive) languages?

- **What problems cannot even be solved by Turing machines that may not halt on all inputs?**
  What problems may not even be formulated into Turing recognizable (i.e., recursively enumerable) languages?
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g.,
\[ 6x^3 - 3x^2 + 24x - 17, \]
has an integral root.

I.e., to test if equation
\[ 6x^3 - 3x^2 + 24x - 17 = 0 \]
has integral solutions.

Because
\[ x = \frac{3}{6}x^2 - \frac{24}{6}x + \frac{17}{6} \]
by dividing both sides with \( 6x^2 \),

we have
\[ -\frac{24}{6} \leq x \leq +\frac{24}{6} \]

because \(|x| \geq 1\).

There is a finite process to decide the question, given such a polynomial.

• enumerate all integers to try, but how?
• enumerate all integers 0, -1, 1, -2, 2, ...
• the process is not infinite because the root should be within the range
  \[ [-\text{c}_{\text{max}} \cdot k, +\text{c}_{\text{max}} \cdot k], \]
  (i.e., \[ [-\frac{24}{6}, +\frac{24}{6}] \])
3.3 Definition of Algorithms

Examples of problems that have Algorithms
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1:

Test if a single-variable polynomial, e.g.,

\[6x^3 - 3x^2 + 24x - 17\]

has an integral root

I.e., to test if equation

\[6x^3 - 3x^2 + 24x - 17 = 0\]

has integral solutions

Because

\[x = \frac{3}{6}x^2 - \frac{24}{6}x + \frac{17}{6}\]

by dividing both sides with \[6x^2\],

we have

\[-\frac{24}{6} \leq x \leq \frac{24}{6}\]

because

\[|x| \geq 1\]
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial,

\[6x^3 - 3x^2 + 24x - 17\]

I.e., to test if equation \([6x^3 - 3x^2 + 24x - 17 = 0]\) has integral solutions

Because \(x = \frac{6}{6} - \frac{24}{6}x + \frac{17}{6}x^2\) by dividing both sides with \(6x^2\), we have

\[-\frac{24}{6} \leq x \leq \frac{24}{6}\]

because \(|x| \geq 1\)

There is a finite process to decide the question, given such a polynomial.

- enumerate all integers to try, but how?
- enumerate all integers 0, -1, 1, -2, 2, ...
- the process is not infinite because the root should be within the range \([-c_{\text{max}} \times k, c_{\text{max}} \times k]\) (i.e., \([-\frac{24}{6} \times 3, \frac{24}{6} \times 3]\))
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., \(6x^3 - 3x^2 + 24x - 17\), has an integral root
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions

or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$

Because $x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}$ by dividing both sides with $6x^2$, 
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions

or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$

Because $x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}$ by dividing both sides with $6x^2$, we have

$$-\frac{24}{6} \times 3 \leq x \leq +\frac{24}{6} \times 3$$
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$

Because $x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}$ by dividing both sides with $6x^2$, we have

$$\frac{-24}{6} \times 3 \leq x \leq \frac{24}{6} \times 3 \quad \text{because} |x| \geq 1$$

There is a finite process to decide the question, given such a polynomial.
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions

or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$

Because $x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}$ by dividing both sides with $6x^2$, we have

$$-\frac{24}{6} \times 3 \leq x \leq +\frac{24}{6} \times 3 \text{ because } |x| \geq 1$$

There is a finite process to decide the question, given such a polynomial.

• enumerate all integers to try,
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., \(6x^3 - 3x^2 + 24x - 17\), has an integral root

I.e., to test if equation \(6x^3 - 3x^2 + 24x - 17 = 0\) has integral solutions

or to test if equation \(6x^3 = 3x^2 - 24x + 17\) holds for some integer \(x\)

Because \(x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}\) by dividing both sides with \(6x^2\), we have

\[-\frac{24}{6} \times 3 \leq x \leq +\frac{24}{6} \times 3 \text{ because } |x| \geq 1\]

There is a finite process to decide the question, given such a polynomial.

- enumerate all integers to try, but how?
Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions

or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$

Because $x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}$ by dividing both sides with $6x^2$, we have

$$-\frac{24}{6} \times 3 \leq x \leq +\frac{24}{6} \times 3 \text{ because } |x| \geq 1$$

There is a finite process to decide the question, given such a polynomial.

- enumerate all integers to try, but how?
- enumerate all integers 0, -1, 1, -2, 2, . . . ,
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., $6x^3 - 3x^2 + 24x - 17$, has an integral root

I.e., to test if equation $6x^3 - 3x^2 + 24x - 17 = 0$ has integral solutions

or to test if equation $6x^3 = 3x^2 - 24x + 17$ holds for some integer $x$

Because $x = 3 - \frac{24}{6x} + \frac{17}{6x^2}$ by dividing both sides with $6x^2$, we have

$$-\frac{24}{6} \times 3 \leq x \leq +\frac{24}{6} \times 3$$

because $|x| \geq 1$

There is a finite process to decide the question, given such a polynomial.

- enumerate all integers to try, but how?
- enumerate all integers 0, -1, 1, -2, 2, . . . ,
- the process is not infinite because
  the root should be within the range $[-\frac{c_{max}}{c_1} \times k, +\frac{c_{max}}{c_1} \times k]$
3.3 Definition of Algorithms

Examples of problems that have Algorithms

Problem 1: Test if a single-variable polynomial, e.g., \(6x^3 - 3x^2 + 24x - 17\), has an integral root

I.e., to test if equation \(6x^3 - 3x^2 + 24x - 17 = 0\) has integral solutions or to test if equation \(6x^3 = 3x^2 - 24x + 17\) holds for some integer \(x\)

Because \(x = \frac{3}{6} - \frac{24}{6x} + \frac{17}{6x^2}\) by dividing both sides with \(6x^2\), we have

\[-\frac{24}{6} \times 3 \leq x \leq +\frac{24}{6} \times 3\] because \(|x| \geq 1\)

There is a finite process to decide the question, given such a polynomial.

- enumerate all integers to try, but how?
- enumerate all integers 0, -1, 1, -2, 2, . . . ,
- the process is not infinite because the root should be within the range \([-\frac{c_{\text{max}}}{c_1} \times k, +\frac{c_{\text{max}}}{c_1} \times k]\) (i.e., \([-\frac{24}{6} \times 3, +\frac{24}{6} \times 3]\))
3.3 Definition of Algorithms

Problems 2: Testing if a given graph is connected (Example 3.23, pages 185-186)

A = \{⟨G⟩| G is connected\}

A is decidable, i.e., there is an algorithm (TM that halts) for it.

On input ⟨G⟩, encoding of G,

1. select the first node and mark it
2. repeat the following until no new nodes are marked
3. for each node in G, mark it if it shares an edge with a marked node
4. if all nodes are mark, accept, otherwise reject

Figure 3.24 (page 186)
Problems 2: Testing if a given graph is connected
(Example 3.23, pages 185-186)

\[ A = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\( A \) is decidable, i.e., there is an algorithm (TM that halts) for it.
Problems 2: Testing if a given graph is connected
(Example 3.23, pages 185-186)

\[ A = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\(A\) is decidable, i.e., there is an algorithm (TM that halts) for it.

On input \(\langle G \rangle\), encoding of \(G\),
Problems 2: Testing if a given graph is connected
(Example 3.23, pages 185-186)

\[ A = \{ \langle G \rangle | \ G \text{ is connected} \} \]

\( A \) is decidable, i.e., there is an algorithm (TM that halts) for it.

On input \( \langle G \rangle \), encoding of \( G \),

1. select the first node and mark it
Problems 2: Testing if a given graph is connected
(Example 3.23, pages 185-186)

\[
A = \{ \langle G \rangle \mid G \text{ is connected} \}
\]

\(A\) is decidable, i.e., there is an algorithm (TM that halts) for it.

On input \(\langle G \rangle\), encoding of \(G\),
1. select the first node and mark it
2. repeat the following until no new nodes are marked
3.3 Definition of Algorithms

Problems 2: Testing if a given graph is connected
(Example 3.23, pages 185-186)

\[ A = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\( A \) is decidable, i.e., there is an algorithm (TM that halts) for it.

On input \( \langle G \rangle \), encoding of \( G \),
1. select the first node and mark it
2. repeat the following until no new nodes are marked
3. for each node in \( G \), mark it if it shares an edge with a marked node
### 3.3 Definition of Algorithms

**Problems 2: Testing if a given graph is connected**  
(Example 3.23, pages 185-186)

\[ A = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\( A \) is decidable, i.e., there is an algorithm (TM that halts) for it.

On input \( \langle G \rangle \), **encoding** of \( G \),

1. select the first node and mark it
2. repeat the following until no new nodes are marked
3. for each node in \( G \), mark it if it shares an edge with a marked node
4. if all nodes are mark, **accept**, otherwise **reject**
Problems 2: Testing if a given graph is connected
(Example 3.23, pages 185-186)

\[
A = \{ \langle G \rangle \mid G \text{ is connected} \}
\]

\(A\) is decidable, i.e., there is an algorithm (TM that halts) for it.

On input \(\langle G \rangle\), encoding of \(G\),
1. select the first node and mark it
2. repeat the following until no new nodes are marked
3. for each node in \(G\), mark it if it shares an edge
   with a marked node
4. if all nodes are marked, accept, otherwise reject

Figure 3.24 (page 186)
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root
e.g.,
\[ 6x^3yz^2 + 3xy^2 - x^3 - 10 \]
• the polynomial may contain multiple variables (e.g., \( x, y, z \))
• it is not possible to get bounds for these variables
• an enumerating process similar to the algorithm for Problem 1 may not halt
• Hilbert10 is not decidable
But it is solvable by a TM (which may not stop)
• enumerate all combinations of integers for \( x, y, z \);
• So Hilbert10 is Turing recognizable (recursively enumerable).
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms
Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[ 6x^3yz^2 + 3yx^2 - x^3 - 10 \]
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[ 6x^3yz^2 + 3xy^2 - x^3 - 10 \]

• the polynomial may contain multiple variables (e.g., \( x, y, z \))
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[
6x^3yz^2 + 3xy^2 - x^3 - 10
\]

- the polynomial may contain multiple variables (e.g., \(x, y, z\))
- it is not possible to get bounds for these variables
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[ 6x^3yz^2 + 3xy^2 - x^3 - 10 \]

- the polynomial may contain multiple variables (e.g., \(x, y, z\))
- it is not possible to get bounds for these variables
- an enumerating process similar to the algorithm for Problem 1 may not halt
Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[ 6x^3yz^2 + 3xy^2 - x^3 - 10 \]

- the polynomial may contain multiple variables (e.g., \( x, y, z \))
- it is not possible to get bounds for these variables
- an enumerating process similar to the algorithm for Problem 1 may not halt
- Hilbert10 is not decidable
Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[ 6x^3yz^2 + 3xy^2 - x^3 - 10 \]

- the polynomial may contain multiple variables (e.g., \( x, y, z \))
- it is not possible to get bounds for these variables
- an enumerating process similar to the algorithm for Problem 1 may not halt
- Hilbert10 is not decidable

But it is solvable by a TM (which may not stop)
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root

\[ e.g., 6x^3yz^2 + 3xy^2 - x^3 - 10 \]

- the polynomial may contain multiple variables (e.g., \( x, y, z \))
- it is not possible to get bounds for these variables
- an enumerating process similar to the algorithm for Problem 1 may not halt
- Hilbert10 is not decidable

But it is solvable by a TM (which may not stop)

- enumerate all combinations of integers for \( x, y, z \);
3.3 Definition of Algorithms

Examples of problems that do not have Algorithms

Problem 3. (Hilbert10) Testing if a polynomial has an integral root
e.g., \(6x^3yz^2 + 3xy^2 - x^3 - 10\)

- the polynomial may contain multiple variables (e.g., \(x, y, z\))
- it is not possible to get bounds for these variables
- an enumerating process similar to the algorithm for Problem 1 may not halt
- Hilbert10 is not decidable

But it is solvable by a TM (which may not stop)

- enumerate all combinations of integers for \(x, y, z\);
- So Hilbert10 is Turing recognizable (recursively enumerable).
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if $L_1, L_2$ are decidable, so is $L_1 \cap L_2$.

Let $M_1$ and $M_2$ are deciders for $L_1$ and $L_2$.

We construct a Turing decider $M$ for $L_1 \cap L_2$ as follows

• on input $w$, $M$ follows the work of $M_1$ on $w$,
  then $M$ follows the work of $M_2$ on $w$,

• $M$ accepts $w$ if both $M_1$ and $M_2$ accepts $w$;
  $M$ rejects $w$ if either $M_1$ or $M_2$ rejects $w$.

$M$ halts on all inputs since $M_1$ and $M_2$ halt on all inputs.
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.
3.3 Definition of Algorithms

**Set operations on decidable, Turing recognizable languages**

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

**How to prove these?**
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if $L_1, L_2$ are decidable, so is $L_1 \cap L_2$. 
Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if \( L_1, L_2 \) are decidable, so is \( L_1 \cap L_2 \).

Let \( M_1 \) and \( M_2 \) are deciders for \( L_1 \) and \( L_2 \).
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if $L_1, L_2$ are decidable, so is $L_1 \cap L_2$.

Let $M_1$ and $M_2$ are deciders for $L_1$ and $L_2$.
We construct a Turing decider $M$ for $L_1 \cap L_2$ as follows
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if $L_1$, $L_2$ are decidable, so is $L_1 \cap L_2$.

Let $M_1$ and $M_2$ are deciders for $L_1$ and $L_2$.

We construct a Turing decider $M$ for $L_1 \cap L_2$ as follows

• on input $w$, $M$ follows the work of $M_1$ on $w$,
  then $M$ follows the work of $M_2$ on $w$. 
3.3 Definition of Algorithms

Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if \( L_1, L_2 \) are decidable, so is \( L_1 \cap L_2 \).

Let \( M_1 \) and \( M_2 \) are deciders for \( L_1 \) and \( L_2 \).
We construct a Turing decider \( M \) for \( L_1 \cap L_2 \) as follows

- on input \( w \), \( M \) follows the work of \( M_1 \) on \( w \),
  then \( M \) follows the work of \( M_2 \) on \( w \),
- \( M \) accepts \( w \) if both \( M_1 \) and \( M_2 \) accepts \( w \);
  \( M \) rejects \( w \) if either \( M_1 \) or \( M_2 \) rejects \( w \).
Set operations on decidable, Turing recognizable languages

The class of decidable languages closed under union, intersection, complementation, concatenation, and star.

How to prove these?

e.g., Prove that if \( L_1, L_2 \) are decidable, so is \( L_1 \cap L_2 \).

Let \( M_1 \) and \( M_2 \) are deciders for \( L_1 \) and \( L_2 \).
We construct a Turing decider \( M \) for \( L_1 \cap L_2 \) as follows

- on input \( w \), \( M \) follows the work of \( M_1 \) on \( w \),
  then \( M \) follows the work of \( M_2 \) on \( w \),
- \( M \) accepts \( w \) if both \( M_1 \) and \( M_2 \) accepts \( w \);
  \( M \) rejects \( w \) if either \( M_1 \) or \( M_2 \) rejects \( w \).

\( M \) halts on all inputs since \( M_1 \) and \( M_2 \) halts on all inputs.
3.3 Definition of Algorithms

E.g., Prove that if \( L \) is decidable, so is \( L^* \).

\[
L^* = \{ \epsilon \} \cup L \cup LL \cup LLL \cup ... \]

\( w \in L^* \iff \exists k, w_1, w_2, ..., w_k \text{ such that } (w = w_1w_2...w_k) \land \forall iw_i \in L \)

How to test the RHS?

Let \( M \) be a decider for \( L \). Construct a decider \( N \) for \( L^* \) as follows:

- on input \( w \),
- \( N \) nondeterministically chooses \( k \);
- \( N \) nondeterministically partitions \( w \) into \( k \) pieces: \( w = w_1w_2...w_k \) (e.g., insert \#s into \( k - 1 \) positions);
- \( N \) follows \( M \) on \( w_i \) for all \( i \);
- \( N \) accepts \( w \) if \( M \) accepts all \( w_i \) (\( N \) rejects \( w \) if \( M \) rejects some \( w_i \)).

But will \( N \) halts?

Yes, if \( k \) is finite. Actually \( 1 \leq k \leq |w| \).
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.  
\((L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)\)
E.g., Prove that if $L$ is decidable, so is $L^*$. 

($L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots$)

$w \in L^*$
3.3 Definition of Algorithms

E.g., Prove that if \( L \) is decidable, so is \( L^* \).

\[
(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)
\]

\[
w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2\ldots w_k)
\]
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)$

$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k$ such that $(w = w_1w_2\ldots w_k) \land (\forall i \ w_i \in L)$
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$$L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots$$

$$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1 w_2 \ldots w_k) \land (\forall i \ w_i \in L)$$

How to test the RHS?
3.3 Definition of Algorithms

E.g., Prove that if \( L \) is decidable, so is \( L^* \).

\[
(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)
\]

\( w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2 \ldots w_k) \land (\forall i \ w_i \in L) \)

How to test the RHS?

Let \( M \) be a decider for \( L \). Construct a decider \( N \) for \( L^* \) as follows:
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$$(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)$$

$$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2 \ldots w_k) \land (\forall i \ w_i \in L)$$

How to test the RHS?

Let $M$ be a decider for $L$. Construct a decider $N$ for $L^*$ as follows:

- on input $w$,
- $N$ nondeterministically chooses $k$;
- for each $i$, run $M$ on $w_i$;
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$$(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)$$

$$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2 \ldots w_k) \land (\forall i \ w_i \in L)$$

How to test the RHS?

Let $M$ be a decider for $L$. Construct a decider $N$ for $L^*$ as follows:

- on input $w$,
- $N$ nondeterministically chooses $k$;
- $N$ nondeterministically partitions $w$ into $k$ pieces: $w = w_1w_2 \ldots w_k$ (e.g., insert $#$s into $k - 1$ positions);
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$$L^* = \{ \epsilon \} \cup L \cup LL \cup LLL \cup \ldots$$

$$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2\ldots w_k) \land (\forall i \ w_i \in L)$$

How to test the RHS?

Let $M$ be a decider for $L$. Construct a decider $N$ for $L^*$ as follows:

- on input $w$,
- $N$ nondeterministically chooses $k$;
- $N$ nondeterministically partitions $w$ into $k$ pieces:
  - $w = w_1w_2\ldots w_k$ (e.g., insert #s into $k - 1$ positions);
- $N$ follows $M$ on $w_i$ for all $i$;
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$L^* = \{ \epsilon \} \cup L \cup LL \cup LLL \cup \ldots$

$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k$ such that $(w = w_1w_2\ldots w_k) \land (\forall i \ w_i \in L)$

How to test the RHS?

Let $M$ be a decider for $L$. Construct a decider $N$ for $L^*$ as follows:

- on input $w$,
- $N$ nondeterministically chooses $k$;
- $N$ nondeterministically partitions $w$ into $k$ pieces:
  $w = w_1w_2\ldots w_k$ (e.g., insert #s into $k - 1$ positions);
- $N$ follows $M$ on $w_i$ for all $i$;
- $N$ accepts $w$ if $M$ accepts all $w_i$ ($N$ rejects $w$ if $M$ rejects some $w_i$)
3.3 Definition of Algorithms

E.g., Prove that if \( L \) is decidable, so is \( L^* \).
\[
(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)
\]

\( w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2\ldots w_k) \land (\forall i \ w_i \in L) \)

How to test the RHS?

Let \( M \) be a decider for \( L \). Construct a decider \( N \) for \( L^* \) as follows:

- on input \( w \),
- \( N \) nondeterministically chooses \( k \);
- \( N \) nondeterministically partitions \( w \) into \( k \) pieces:
  \( w = w_1w_2\ldots w_k \) (e.g., insert \#s into \( k - 1 \) positions);
- \( N \) follows \( M \) on \( w_i \) for all \( i \);
- \( N \) accepts \( w \) if \( M \) accepts all \( w_i \) (\( N \) rejects \( w \) if \( M \) rejects some \( w_i \))

But will \( N \) halts?
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$$(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)$$

$$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2\ldots w_k) \land (\forall i \; w_i \in L)$$

How to test the RHS?

Let $M$ be a decider for $L$. Construct a decider $N$ for $L^*$ as follows:

- on input $w$,
- $N$ nondeterministically chooses $k$;
- $N$ nondeterministically partitions $w$ into $k$ pieces:
  $w = w_1w_2\ldots w_k$ (e.g., insert $#$s into $k - 1$ positions);
- $N$ follows $M$ on $w_i$ for all $i$;
- $N$ accepts $w$ if $M$ accepts all $w_i$ ($N$ rejects $w$ if $M$ rejects some $w_i$)

But will $N$ halts? Yes, if $k$ is finite.
3.3 Definition of Algorithms

E.g., Prove that if $L$ is decidable, so is $L^*$.

$$(L^* = \{\epsilon\} \cup L \cup LL \cup LLL \cup \ldots)$$

$$w \in L^* \iff \exists k, w_1, w_2, \ldots, w_k \text{ such that } (w = w_1w_2\ldots w_k) \land (\forall i \ w_i \in L)$$

How to test the RHS?

Let $M$ be a decider for $L$. Construct a decider $N$ for $L^*$ as follows:

- on input $w$,
- $N$ nondeterministically chooses $k$;
- $N$ nondeterministically partitions $w$ into $k$ pieces:
  $$w = w_1w_2\ldots w_k$$
  (e.g., insert #s into $k - 1$ positions);
- $N$ follows $M$ on $w_i$ for all $i$;
- $N$ accepts $w$ if $M$ accepts all $w_i$ ($N$ rejects $w$ if $M$ rejects some $w_i$)

But will $N$ halts? Yes, if $k$ is finite. Actually $1 \leq k \leq |w|$. !
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?

- reversing answers no longer work if a TM may not halt.

Claim 1. If both $L$ and $L'$ are Turing recognizable, then $L$ is decidable.

Proof?

Claim 2. There are languages that are NOT even Turing recognizable.

Proof: Hilbert10 is not Turing recognizable.

Otherwise, because Hilbert10 is Turing recognizable, by Claim 1, Hilbert10 is decidable.

But we know it is not decidable (contradiction).
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.
The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.

**Claim 1.** If both $L$ and $\overline{L}$ are Turing recognizable, then $L$ is decidable.
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.

Claim 1. If both $L$ and $\overline{L}$ are Turing recognizable, then $L$ is decidable.

Proof?
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.

Claim 1. If both $L$ and $\overline{L}$ are Turing recognizable, then $L$ is decidable.

Proof?

Claim 2. There are languages that are NOT even Turing recognizable.
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.

**Claim 1.** If both $L$ and $\overline{L}$ are Turing recognizable, then $L$ is decidable.

*Proof?*

**Claim 2.** There are languages that are NOT even Turing recognizable.

*Proof: Hilbert10 is not Turing recognizable.*
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.

**Claim 1.** If both \( L \) and \( \overline{L} \) are Turing recognizable, then \( L \) is decidable.

*Proof?*

**Claim 2.** There are languages that are NOT even Turing recognizable.

*Proof: Hilbert10 is not Turing recognizable.*

Otherwise, because Hilbert10 is Turing recognizable, by Claim 1, Hilbert10 is decidable.
3.3 Definition of Algorithms

The class of Turing recognizable languages closed under union, intersection, concatenation, and star.

Why not closed under complementation?
- reversing answers no longer work if a TM may not halt.

Claim 1. If both $L$ and $\overline{L}$ are Turing recognizable, then $L$ is decidable.

Proof?

Claim 2. There are languages that are NOT even Turing recognizable.

Proof: Hilbert10 is not Turing recognizable.

Otherwise, because Hilbert10 is Turing recognizable, by Claim 1, Hilbert10 is decidable.

But we know it is not decidable (contradiction).
Chapter 4. Decidability
Chapter 4. Decidability

Figure: 4.1 The Chomsky Hierarchy

- FA: finite state automaton
- RE: regular expression
- RG: regular grammar
- CFG: context-free grammar
- PDA: pushdown automaton
- LBA: linear-bounded automaton
- CSG: context-sensitive grammar
- TM: Turing machine
Chapter 4. Decidability

4.1 Decidable Languages
- How to prove a language is decidable
- Decidable languages concerning computation models

4.2 Undecidable Languages
- How to prove a language is not decidable
- The first language proved to be undecidable: Halting
- Other undecidable languages
Chapter 4. Decidability

4.1 Decidable Languages

- How to prove a language is decidable
- decidable languages concerning computation models;
Chapter 4. Decidability

- 4.1 Decidable Languages
  - How to prove a language is decidable
  - Decidable languages concerning computation models;

- 4.2 Undecidable Languages
  - How to prove a language is not decidable
  - The first language proved to be undecidable: Halting
  - Other undecidable languages
Chapter 4. Decidability

4.1 Decidable Languages

• To prove a language is decidable, an algorithm suffices.

• We translate one language to another with "simulation techniques".

Chapter 4. Decidability

4.1 Decidable Languages
4.1 Decidable Languages

• To prove a language is decidable, an algorithm suffices.
4.1 Decidable Languages

- To prove a language is decidable, an algorithm suffices.
- We translate one language to another with “simulation techniques”.

Chapter 4. Decidability

4.1 Decidable Languages

Almost all languages/problems we have seen are decidable.

Example 1: language $\text{Prime} = \{p \in \{0, 1\}^* : p$ encodes a prime number $\}$.

Example 2: language $\text{Sorted List} = \{x_1 \# \ldots \# x_n : n \geq 1, x_1 \preceq \ldots \preceq x_n, x_i \in \{0, 1\}^*, 1 \leq i \leq n\}$ where $\preceq$ is the lexicographical order.

Example 3: TSP problem $\text{L}_{TSP} = \{\langle G, k \rangle : \text{weighted graph } G \text{ has a circular tour of weight } \leq k\}$. 
Chapter 4. Decidability

4.1 Decidable Languages
Chapter 4. Decidability

4.1 Decidable Languages

Almost all languages/problems we have seen are decidable.
Chapter 4. Decidability

4.1 Decidable Languages

Almost all languages/problems we have seen are decidable.

Example 1: language \textsc{Prime}

\textsc{Prime} = \{ p : p \in \{0, 1\}^*, p \text{ encodes a prime number} \}
4.1 Decidable Languages

Almost all languages/problems we have seen are decidable.

Example 1: language PRIME

\[ \text{PRIME} = \{ p : p \in \{0, 1\}^*, \text{ p encodes a prime number } \} \]

Example 2: language SORTED LIST

\[ L_{SL} = \{ x_1 \# \ldots \# x_n : n \geq 1, x_1 \leq \cdots \leq x_n, x_i \in \{0, 1\}^*, 1 \leq i \leq n \} \]

where \( \leq \) is the lexicographical order
Chapter 4. Decidability

4.1 Decidable Languages

Almost all languages/problems we have seen are decidable.

Example 1: language PRIME

PRIME = \{ p : p \in \{0, 1\}^*, p \text{ encodes a prime number} \}

Example 2: language SORTED LIST

L_{SL} = \{ x_1 \# \ldots \# x_n : n \geq 1, x_1 \leq \ldots \leq x_n, x_i \in \{0, 1\}^*, 1 \leq i \leq n \}

where \leq is the lexicographical order

Example 3: TSP problem

L_{TSP} = \{ \langle G, k \rangle : \text{weighted graph } G \text{ has a circular tour of weight } \leq k \}
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions
CFL = CFG = PDA

There are two techniques to prove a language $L$ is decidable:
1. directly design a decider (algorithm) to recognize $L$;
2. use the fact that another language $A$ is decidable; transform the recognition of $L$ to recognition of $A$; $L \leq A$.

The transformation + the decider for $A$ yields a decider for $L$. 
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages $\equiv$ DFA $\equiv$ NFA $\equiv$ Regular expressions
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions

CFL = CFG = PDA
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions

CFL = CFG = PDA

TM
We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions
CFL = CFG = PDA
TM

There are two techniques to prove a language $L$ is decidable
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions
CFL = CFG = PDA
TM

There are two techniques to prove a language $L$ is decidable
1. directly design a decider (algorithm) to recognize $L$;
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions
CFL = CFG = PDA
TM

There are two techniques to prove a language $L$ is decidable:
1. directly design a decider (algorithm) to recognize $L$;
2. use the fact that another language $A$ is decidable;
4.1 Decidable Languages

We will examine decidable languages that are formulated from the machine models and their equivalences.

Regular languages = DFA = NFA = Regular expressions
CFL = CFG = PDA
TM

There are two techniques to prove a language $L$ is decidable

1. directly design a decider (algorithm) to recognize $L$;
2. use the fact that another language $A$ is decidable;
   transform the recognition of $L$ to recognition of $A$; $L \leq A$.
   the transformation + the decider for $A$ yields a decider for $L$. 
4.1 Decidable Languages

Decidable problems concerning regular languages

Accepting problem for DFAs: testing if a given DFA accepts a given string

A DFA = \{⟨B,w⟩ | B is a DFA that accepts string w\}

Theorem 4.1

A DFA is a decidable language.

Note: To prove a language is decidable, it suffices to construct a TM (that halts on all inputs) to recognize the language.
Decidable problems concerning regular languages
4.1 Decidable Languages

Decidable problems concerning regular languages

Accepting problem for DFAs: testing if a given DFA accepts a given string

$$A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \}$$
4.1 Decidable Languages

Decidable problems concerning regular languages

Accepting problem for DFAs: testing if a given DFA accepts a given string

\[ A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \} \]

**Theorem 4.1** \( A_{DFA} \) is a decidable language.
Decidable problems concerning regular languages

Accepting problem for DFAs: testing if a given DFA accepts a given string

$$A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \}$$

**Theorem 4.1** $A_{DFA}$ is a decidable language.

**Note**: To prove a language is decidable, it suffices to construct a TM (that halts on all inputs) to recognize the language.
4.1 Decidable Languages

Theorem 4.1

A DFA is a decidable language.

proof idea:

• Construct TM $M_1$, on input $\langle B, w \rangle$ to follow the work of $B$ on $w$.
  • keep track of $B$'s current state and position on $w$.

Some details:

• how is $B$ encoded?
  • how to keep track of $B$ states?
  • how to keep track of positions on $w$?
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.

*proof idea:*

- Construct TM $M_1$, on input $⟨B,w⟩$, to follow the work of $B$ on $w$.
- Keep track of $B$'s current state and position on $w$.
- $M_1$ accepts $⟨B,w⟩$ if $B$ accepts $w$.

Some details:

- How is $B$ encoded?
- How to keep track of $B$ states?
- How to keep track of positions on $w$?
4.1 Decidable Languages

**Theorem 4.1** \( A_{DFA} \) is a decidable language.

**proof idea:**

Construct TM \( M_1 \), on input \( \langle B, w \rangle \), to follow the work of \( B \) on \( w \).
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.

**proof idea:**

Construct TM $M_1$, on input $\langle B, w \rangle$, to follow the work of $B$ on $w$.

- keep track of $B$’s current state and position on $w$. 

Some details:
- how is $B$ encoded?
- how to keep track of $B$ states?
- how to keep track of positions on $w$?
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.

**proof idea:**

Construct TM $M_1$, on input $\langle B, w \rangle$, to follow the work of $B$ on $w$.

- keep track of $B$’s current state and position on $w$.
- $M_1$ accepts $\langle B, w \rangle$ iff $B$ accepts $w$
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.

**proof idea:**

Construct TM $M_1$, on input $\langle B, w \rangle$, to follow the work of $B$ on $w$.

- keep track of $B$'s current state and position on $w$.
- $M_1$ accepts $\langle B, w \rangle$ iff $B$ accepts $w$

**Some details:**
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.

**proof idea:**

Construct TM $M_1$, on input $\langle B, w \rangle$, to follow the work of $B$ on $w$.

- keep track of $B$’s current state and position on $w$.
- $M_1$ accepts $\langle B, w \rangle$ iff $B$ accepts $w$

**Some details:**

- how is $B$ encoded?
4.1 Decidable Languages

**Theorem 4.1** \( A_{DFA} \) is a decidable language.

**proof idea:**

Construct TM \( M_1 \), on input \( \langle B, w \rangle \), to follow the work of \( B \) on \( w \).

- keep track of \( B \)'s current state and position on \( w \).
- \( M_1 \) accepts \( \langle B, w \rangle \) iff \( B \) accepts \( w \)

**Some details:**

- how is \( B \) encoded?
- how to keep track of \( B \) states?
4.1 Decidable Languages

**Theorem 4.1** $A_{DFA}$ is a decidable language.

**proof idea:**

Construct TM $M_1$, on input $⟨B, w⟩$, to follow the work of $B$ on $w$.

- keep track of $B$’s current state and position on $w$.

- $M_1$ accepts $⟨B, w⟩$ iff $B$ accepts $w$.

**Some details:**

- how is $B$ encoded?
- how to keep track of $B$ states?
- how to keep track of positions on $w$?
4.1 Decidable Languages

Similarly, \( A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \)

Theorem 4.2
\( A_{NFA} \) is a decidable language.

Proof ideas:
1. Use an NTM \( N \) to simulate NFA \( B \) on \( w \),
   • \( N \) accepts \( \langle B, w \rangle \) iff \( B \) accepts \( w \). 
2. Alternatively, \( N \) converts \( B \) an equivalent DFA \( B' \),
   • then \( N \) follows TM \( M_1 \) (constructed in Theorem 4.1) on \( \langle B', w \rangle \),
   • \( N \) accepts \( \langle B, w \rangle \) iff \( M_1 \) accepts \( \langle B', w \rangle \).
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem 4.2** \( A_{NFA} \) is a decidable language.
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem 4.2** \( A_{NFA} \) is a decidable language.

**proof ideas**
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem 4.2** \( A_{NFA} \) is a decidable language.

**proof ideas**

1. Use an NTM \( N \) to simulate NFA \( B \) on \( w \),
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem 4.2** \( A_{NFA} \) is a decidable language.

**proof ideas**

1. Use an NTM \( N \) to simulate NFA \( B \) on \( w \),
   - \( N \) accepts \( \langle B, w \rangle \) iff \( B \) accepts \( w \).
4.1 Decidable Languages

Similarly

$$A_{NFA} = \{ \langle B, w \rangle | B \text{ is an NFA that accepts string } w \}$$

**Theorem 4.2** $A_{NFA}$ is a decidable language.

**proof ideas**

1. Use an NTM $N$ to simulate NFA $B$ on $w$,
   - $N$ accepts $\langle B, w \rangle$ iff $B$ accepts $w$.
2. Alternatively, $N$ converts $B$ an equivalent DFA $B'$,
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem 4.2** \( A_{NFA} \) is a decidable language.

**proof ideas**

1. Use an NTM \( N \) to simulate NFA \( B \) on \( w \),
   - \( N \) accepts \( \langle B, w \rangle \) iff \( B \) accepts \( w \).

2. Alternatively, \( N \) converts \( B \) an equivalent DFA \( B' \),
   - then \( N \) follows TM \( M_1 \) (constructed in Theorem 4.1) on \( \langle B', w \rangle \),
4.1 Decidable Languages

Similarly

\[ A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem 4.2** \( A_{NFA} \) is a decidable language.

**proof ideas**

1. Use an NTM \( N \) to simulate NFA \( B \) on \( w \),
   - \( N \) accepts \( \langle B, w \rangle \) iff \( B \) accepts \( w \).

2. Alternatively, \( N \) converts \( B \) an equivalent DFA \( B' \),
   - then \( N \) follows TM \( M_1 \) (constructed in Theorem 4.1) on \( \langle B', w \rangle \),
   - \( N \) accepts \( \langle B, w \rangle \) iff \( M_1 \) accepts \( \langle B', w \rangle \).
4.1 Decidable Languages

**Theorem 4.3**

A $\text{REX}$ is a decidable language.

**Proof Idea**

Construct a TM $P$ to decide $\text{REX}$:

- On input $\langle R, w \rangle$;
- $P$ converts $R$ to an equivalent NFA $B$ using the finite steps provided in Theorem 1.54;
- $P$ runs $N$ (Theorem 4.2) on input $\langle B, w \rangle$;
- $P$ accepts $\langle R, w \rangle$ iff $N$ accepts $\langle B, w \rangle$. 
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** \( A_{REX} \) is a decidable language.
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** \( A_{REX} \) is a decidable language.

**proof idea?**
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** $A_{REX}$ is a decidable language.

**proof idea?**

Construct a TM $P$ to decide $A_{REX}$:
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** \( A_{REX} \) is a decidable language.

**proof idea**?

Construct a TM \( P \) to decide \( A_{REX} \):

On input \( \langle R, w \rangle \):
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** \( A_{REX} \) is a decidable language.

**proof idea?**

Construct a TM \( P \) to decide \( A_{REX} \):

On input \( \langle R, w \rangle \):

- \( P \) convert \( R \) to an equivalent NFA \( B \) using the finite steps provided in theorem 1.54
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** \( A_{REX} \) is a decidable language.

**proof idea?**

Construct a TM \( P \) to decide \( A_{REX} \):

- On input \( \langle R, w \rangle \):
  - \( P \) convert \( R \) to an equivalent NFA \( B \) using the finite steps provided in theorem 1.54
  - \( P \) runs \( N \) (theorem 4.2) on input \( \langle B, w \rangle \)
4.1 Decidable Languages

\[ A_{REX} = \{ \langle R, w \rangle \mid \text{regular expression } R \text{ generates string } w \} \]

**Theorem 4.3** \( A_{REX} \) is a decidable language.

**proof idea?**

Construct a TM \( P \) to decide \( A_{REX} \):

On input \( \langle R, w \rangle \):
- \( P \) convert \( R \) to an equivalent NFA \( B \) using the finite steps provided in theorem 1.54
- \( P \) runs \( N \) (theorem 4.2) on input \( \langle B, w \rangle \)
- \( P \) accepts \( \langle R, w \rangle \) iff \( N \) accepts \( \langle B, w \rangle \)
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

Theorem 4.4

\text{E DFA} \text{ is a decidable language.}

Proof idea?

To determine if there is a path from the initial state to an accept state.

Design a TM \( T \) for this purpose:

1. Mark the start state of \( A \).
2. Repeat until no new states get marked:
   3. Mark any state that has a transition coming to it from a state that has been marked;
   4. If no accept state is marked, \( T \) accept, otherwise rejects.
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

\[ E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \]
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**Proof idea**?
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**proof idea**?

To determine if there is path from the initial state to a accept state.
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**proof idea**

To determine if there is path from the initial state to a accept state.

Design a TM \( T \) for this purpose
Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**proof idea**

To determine if there is path from the initial state to an accept state.

Design a TM \( T \) for this purpose

On the input \( \langle A \rangle \);
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{\text{DFA}} \) is a decidable language.

**proof idea**

To determine if there is path from the initial state to a accept state.

Design a TM \( T \) for this purpose

On the input \( \langle A \rangle \);
1. Mark the start state of \( A \).
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**proof idea**

To determine if there is path from the initial state to a accept state.

Design a TM \( T \) for this purpose

On the input \( \langle A \rangle \);
1. Mark the start state of \( A \).
2. Repeat until no new states get marked:
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**proof idea**

To determine if there is path from the initial state to an accept state.

Design a TM \( T \) for this purpose

On the input \( \langle A \rangle \):
1. Mark the start state of \( A \).
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming to it from a state that has been marked;
4.1 Decidable Languages

Testing if a DFA accepts the empty language (rejects all strings)

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

**Theorem 4.4** \( E_{DFA} \) is a decidable language.

**proof idea**

To determine if there is path from the initial state to a accept state.

Design a TM \( T \) for this purpose

On the input \( \langle A \rangle \);
1. Mark the start state of \( A \).
2. Repeat until no new states get marked:
   3. Mark any state that has a transition coming to it from a state that has been marked;
4. If no accept state is marked, \( T \) accept, otherwise rejects.
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ \text{EQ}_{\text{DFA}} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5**

\( \text{EQ}_{\text{DFA}} \) is a decidable language.

**proof idea:** To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[ L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset \iff (L(A) \cap L(B)) \cup (L(B) \cap L(A)) = \emptyset \]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( \text{EQ}_{\text{DFA}} \) as follows:

1. On input \( \langle A, B \rangle \),
2. produces a DFA \( C \) for language \( (L(A) \cap L(B)) \cup (L(B) \cap L(A)) \),
3. follows TM \( T \) (of theorem 4.4) on \( \langle C \rangle \);
4. accepts iff \( T \) accepts.
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{\text{DFA}} \) is a decidable language.
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**

1. On input \( \langle A, B \rangle \),
2. construct a DFA \( C \) for language \((L(A) \cap L(B)) \cup (L(B) \cap L(A))\);
3. follow TM \( T \) (of theorem 4.4) on \( \langle C \rangle \);
4. accept if \( T \) accepts.
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**
To relate testing \( L(A) = L(B) \) to a empty language testing. How?
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**
To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[ L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset \]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( EQ_{DFA} \) as follows:
1. On input \( \langle A, B \rangle \),
2. produces a DFA \( C \) for language \( (L(A) \cap L(B)) \cup (L(B) \cap L(A)) \)
3. follows TM \( T \) (of theorem 4.4) on \( \langle C \rangle \);
4. accepts \( \text{iff} \) \( T \) accepts.
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**

To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[
L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset
\]

\[
\iff (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset
\]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**

To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[ L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset \]

\[ \iff (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset \]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( EQ_{DFA} \) as follows:
4.1 Decidable Languages

Testing if two DFAs are equivalent

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

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**
To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[
L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset
\]

\[
\iff (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset
\]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( EQ_{DFA} \) as follows:
1. On input \( \langle A, B \rangle \),
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**
To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[ L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset \]

\[ \iff (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset \]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( EQ_{DFA} \) as follows:

1. On input \( \langle A, B \rangle \),
2. produces a DFA \( C \) for language \( (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) \)
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**
To relate testing \( L(A) = L(B) \) to a empty language testing. How?

\[ L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset \]

\[ \iff (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset \]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( EQ_{DFA} \) as follows:
1. On input \( \langle A, B \rangle \),
2. produces a DFA \( C \) for language \( (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) \)
3. follows TM \( T \) (of theorem 4.4) on \( \langle C \rangle \);
4.1 Decidable Languages

Testing if two DFAs are equivalent

\[ EQ_{DFA} = \{ \langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B) \} \]

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

**proof idea:**
To relate testing \( L(A) = L(B) \) to an empty language testing. How?

\[
L(A) = L(B) \iff (L(A) - L(B)) \cup (L(B) - L(A)) = \emptyset
\]
\[
\iff (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset
\]

All the operations: union, intersection, and complementation can be done in finite steps and still result in regular languages.

Construct a TM \( M \) to decide for language \( EQ_{DFA} \) as follows:
1. On input \( \langle A, B \rangle \),
2. produces a DFA \( C \) for language \( (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) \)
3. follows TM \( T \) (of theorem 4.4) on \( \langle C \rangle \);
4. accepts iff \( T \) accepts.
4.1 Decidable Languages

Decidable problems concerning context-free languages

A CFG \( G = \{ \langle G, w \rangle | G \text{ is a CFG that generates string } w \} \),

Theorem 4.7

A CFG is a decidable language.

Proof idea:

Use Chomsky normal form (\( S_0 \rightarrow \epsilon, X \rightarrow YZ, X \rightarrow a \))

(1) can check if \( w = \epsilon \)

(2) let length \( |w| = l \), assume there is a derivation,

- how many steps needed to derive \( w \)?
- each step use of rule \( X \rightarrow YZ \) generates at least one symbol
- like \( X \Rightarrow YZ \Rightarrow aZ \), two steps

Derivation of one additional symbol needs two steps except the last one.

So totally \( 2l - 1 \) steps.

Try all derivations of \( 2l - 1 \) steps.

how many of them?

\(|R| \geq 2^{l-1}, \text{ finite!} \)
4.1 Decidable Languages

Decidable problems concerning context-free languages

Theorem 4.7

$A_{CFG} = \{ \langle G, w \rangle | G \text{ is a CFG that generates string } w \}$

Proof idea:

1. Can check if $w = \epsilon$
2. Let length $|w| = l$, assume there is a derivation,
   - how many steps needed to derive $w$?
   - each step use of rule $X \rightarrow YZ, X \rightarrow a$ generates at least one symbol
   - like $X \Rightarrow YZ \Rightarrow aZ$, two steps
   - Derivation of one additional symbol needs two steps except the last one.
   - So totally $2l - 1$ steps
3. Try all derivations of $2l - 1$ steps, finite!
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{\text{CFG}} \) is a decidable language.
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{CFG} \) is a decidable language.

**Proof idea:**
Use Chomsky normal form \((S_0 \to \epsilon, X \to YZ, X \to a)\)
Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{CFG} \) is a decidable language.

**Proof idea:**
Use Chomsky normal form \((S_0 \rightarrow \epsilon, X \rightarrow YZ, X \rightarrow a)\)

(1) can check if \( w = \epsilon \)
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{CFG} \) is a decidable language.

**Proof idea:**
Use Chomsky normal form \((S_0 \rightarrow \epsilon, X \rightarrow YZ, X \rightarrow a)\)

(1) can check if \( w = \epsilon \)

(2) let length \( |w| = l \), assume there is a derivation,
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{CFG} \) is a decidable language.

**Proof idea:**
Use Chomsky normal form \((S_0 \rightarrow \epsilon, X \rightarrow YZ, X \rightarrow a)\)

1. can check if \( w = \epsilon \)

2. let length \(|w| = l\), assume there is a derivation,
   - how many steps needed to derive \( w \)?
   - each step use of rule \( X \rightarrow YZ \) generates at least one symbol
   - like \( X \Rightarrow YZ \Rightarrow aZ \), two steps
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{CFG} \) is a decidable language.

**Proof idea:**
Use Chomsky normal form \((S_0 \to \epsilon, X \to YZ, X \to a)\)

(1) can check if \( w = \epsilon \)

(2) let length \( |w| = l \), assume there is a derivation,
   - **how many steps needed to derive** \( w \)?
   - each step use of rule \( X \to YZ \) generates at least one symbol
   - like \( X \Rightarrow YZ \Rightarrow aZ \), two steps

Derivation of one additional symbol needs two steps except the last one.
So totally \( 2l - 1 \) steps
4.1 Decidable Languages

Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

Theorem 4.7 \( A_{CFG} \) is a decidable language.

Proof idea:
Use Chomsky normal form (\( S_0 \rightarrow \epsilon, X \rightarrow YZ, X \rightarrow a \))

(1) can check if \( w = \epsilon \)

(2) let length \( |w| = l \), assume there is a derivation,
  - how many steps needed to derive \( w \)?
  - each step use of rule \( X \rightarrow YZ \) generates at least one symbol
  - like \( X \Rightarrow YZ \Rightarrow aZ \), two steps

Derivation of one additional symbol needs two steps except the last one. So totally \( 2l - 1 \) steps

Try all derivations of \( 2l - 1 \) steps.
  - how many of them?
Decidable problems concerning context-free languages

\[ A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]

**Theorem 4.7** \( A_{CFG} \) is a decidable language.

**Proof idea:**
Use Chomsky normal form \((S_0 \rightarrow \epsilon, X \rightarrow YZ, X \rightarrow a)\)

1. Can check if \( w = \epsilon \)

2. Let length \(|w| = l\), assume there is a derivation,
   - how many steps needed to derive \( w \)?
   - each step use of rule \( X \rightarrow YZ \) generates at least one symbol
   - like \( X \Rightarrow YZ \Rightarrow aZ \), two steps

Derivation of one additional symbol needs two steps except the last one.
So totally \( 2l - 1 \) steps

Try all derivations of \( 2l - 1 \) steps.
   - how many of them? \(|R|^{2l-1}\), finite!
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ \text{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \emptyset \} \]

Theorem 4.8

\text{CFG} is a decidable language.

Proof idea:

Design a TM as follows such that:

- On input \( \langle G \rangle \), encoding of \( G \),
  - mark all terminal symbols in \( G \),
  - repeat until no new variables get marked:
    - mark any variable \( A \) if \( A \rightarrow \alpha \) is a rule and all symbols in \( \alpha \) has been marked.
  - if the start variable is not marked, accept; otherwise reject.
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \phi \} \]
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \phi \} \]

**Theorem 4.8** \( E_{CFG} \) is a decidable language.
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \phi \} \]

**Theorem 4.8** \( E_{CFG} \) is a decidable language.

**Proof idea:**
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

$$E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \phi \}$$

**Theorem 4.8** $E_{CFG}$ is a decidable language.

**Proof idea:**
Design a TM as follows such that

On input $\langle G \rangle$, encoding of $G$,
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \phi \} \]

**Theorem 4.8** \( E_{CFG} \) is a decidable language.

**Proof idea:**

Design a TM as follows such that

On input \( \langle G \rangle \), encoding of \( G \),

- mark all terminal symbols in \( G \),

- repeat until no new variables get marked:
  - mark any variable \( A \) if \( A \rightarrow \alpha \) is a rule and all symbols in \( \alpha \) has been marked.

- if the start variable is not marked, accept; otherwise reject.
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \]

**Theorem 4.8** \( E_{CFG} \) is a decidable language.

**Proof idea:**

Design a TM as follows such that

- On input \( \langle G \rangle \), encoding of \( G \),
  - mark all terminal symbols in \( G \),
  - repeat until no new variables get marked:

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \]
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \phi \} \]

**Theorem 4.8** \( E_{CFG} \) is a decidable language.

**Proof idea:**

Design a TM as follows such that

On input \( \langle G \rangle \), encoding of \( G \),

- mark all terminal symbols in \( G \),
- repeat until no new variables get marked:
  - mark any variable \( A \) if \( A \rightarrow \alpha \) is a rule
    and all symbols in \( \alpha \) has been marked.
4.1 Decidable Languages

Also we have the problem of testing if a CFG generates the empty language.

\[ E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \]

**Theorem 4.8** \( E_{CFG} \) is a decidable language.

**Proof idea:**

Design a TM as follows such that

On input \( \langle G \rangle \), encoding of \( G \),

- mark all terminal symbols in \( G \),
- repeat until no new variables get marked:
  - mark any variable \( A \) if \( A \rightarrow \alpha \) is a rule
    and all symbols in \( \alpha \) has been marked.
- if the start variable is not marked, accept; otherwise reject.
4.1 Decidable Languages

We would hope the following language is also decidable, but it is not.

$$\text{EQ}_{\text{CFG}} = \{ \langle G, H \rangle \mid G, H \text{ are CFGs and } L(G) = L(H) \}$$

The technique used to prove Theorem 4.5 is not applicable to testing if two CFLs are equal.

The class of CFLs are not closed under intersection or complement.
4.1 Decidable Languages

We would hope the following language is also decidable, but it is not.

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

We would hope the following language is also decidable, but it is not.

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

The technique used to prove Theorem 4.5 is not applicable to testing if two CFLs are equal.
4.1 Decidable Languages

We would hope the following language is also decidable, but it is not.

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

The technique used to prove Theorem 4.5 is not applicable to testing if two CFLs are equal.

The class of CFLs are not closed under intersection or complement.
4.1 Decidable Languages

Theorem 4.9
Every CFL is decidable.

Proof idea:
Let $G$ be the CFG for the language under consideration. Construct a TM to simulate the work of $G$ as follows:

- On input $w$,
  - simulates the TM of Theorem 4.7 on $⟨G, w⟩$,
  - accepts iff the TM of Theorem 4.7 accepts.
Theorem 4.9 Every CFL is decidable.
4.1 Decidable Languages

**Theorem 4.9** Every CFL is decidable.

**Proof idea:**

Let $G$ be the CFG for the language under consideration. Construct a TM to simulate the work of $G$ as follows:

- On input $w$,
  - simulate the TM of Theorem 4.7 on $\langle G, w \rangle$,
  - accepts if the TM of Theorem 4.7 accepts.
4.1 Decidable Languages

**Theorem 4.9** Every CFL is decidable.

**Proof idea:**

Let $G$ be the CFG for the language under consideration. Construct a TM to simulate the work of $G$ as follows:
4.1 Decidable Languages

**Theorem 4.9** Every CFL is decidable.

**Proof idea:**

Let $G$ be the CFG for the language under consideration. Construct a TM to simulate the work of $G$ as follows:

On input $w$, 

4.1 Decidable Languages

**Theorem 4.9** Every CFL is decidable.

**Proof idea:**

Let $G$ be the CFG for the language under consideration. Construct a TM to simulate the work of $G$ as follows:

On input $w$,

- simulates the TM of Theorem 4.7 on $\langle G, w \rangle$
4.1 Decidable Languages

**Theorem 4.9** Every CFL is decidable.

**Proof idea:**

Let $G$ be the CFG for the language under consideration. Construct a TM to simulate the work of $G$ as follows:

On input $w$,
- simulates the TM of Theorem 4.7 on $\langle G, w \rangle$
- accepts iff the TM of Theorem 4.7 accepts.
4.1 Decidable Languages

The relationship among classes of languages

\[ \text{regular} \subset \text{context-free} \subset \text{decidable} \subset \text{Turing-recognizable} \]

- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved Hilber 10 is undecidable.

There are two techniques to prove a language \( L \) is not decidable

1. directly prove that \( L \) is not decidable;
2. use the fact that another language \( B \) is not decidable, transform the recognition of \( B \) to recognition of \( L \), i.e., \( B \leq L \) , a decider for \( L \) + transformation would yield a decider for \( B \), contradict.
4.1 Decidable Languages

The relationship among classes of languages

- regular $\subset$ context-free $\subset$ decidable $\subset$ Turing-recognizable
- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved Hilbert's 10th problem is undecidable.

There are two techniques to prove a language $L$ is not decidable:
1. directly prove that $L$ is not decidable;
2. use the fact that another language $B$ is not decidable, transform the recognition of $B$ to recognition of $L$, i.e., $B \leq L$, a decider for $L$ + transformation would yield a decider for $B$, contradict.
4.1 Decidable Languages

The relationship among classes of languages

regular $\subset$ context-free $\subset$ decidable $\subset$ Turing-recognizable
4.1 Decidable Languages

The relationship among classes of languages

regular ⊂ context-free ⊂ decidable ⊂ Turing-recognizable

• the containments are all proper;
4.1 Decidable Languages

The relationship among classes of languages

regular $\subset$ context-free $\subset$ decidable $\subset$ Turing-recognizable

- the containments are all proper;
- we know some specific languages that made each containment proper.

There are two techniques to prove a language $L$ is not decidable
1. directly prove that $L$ is not decidable;
2. use the fact that another language $B$ is not decidable, transform the recognition of $B$ to recognition of $L$, i.e., $B \leq L$, a decider for $L$ + transformation would yield a decider for $B$, contradict.
4.1 Decidable Languages

The relationship among classes of languages

regular $\subset$ context-free $\subset$ decidable $\subset$ Turing-recognizable

- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved $Hilber10$ is undecidable.
The relationship among classes of languages

regular ⊂ context-free ⊂ decidable ⊂ Turing-recognizable

- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved Hilber10 is undecidable.

There are two techniques to prove a language \( L \) is not decidable

1. directly prove that \( L \) is not decidable;
2. use the fact that another language \( B \) is not decidable, transform the recognition of \( B \) to recognition of \( L \), i.e., \( B \leq L \), a decider for \( L \) + transformation would yield a decider for \( B \), contradict.
The relationship among classes of languages

regular \subset \text{context-free} \subset \text{decidable} \subset \text{Turing-recognizable}

- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved \textit{Hilbert10} is undecidable.

There are two techniques to prove a language \( L \) is not decidable

1. directly prove that \( L \) is not decidable;
4.1 Decidable Languages

The relationship among classes of languages

regular \(\subset\) context-free \(\subset\) decidable \(\subset\) Turing-recognizable

- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved Hilber10 is undecidable.

There are two techniques to prove a language \(L\) is not decidable
1. directly prove that \(L\) is not decidable;
2. use the fact that another language \(B\) is not decidable,
4.1 Decidable Languages

The relationship among classes of languages

regular $\subset$ context-free $\subset$ decidable $\subset$ Turing-recognizable

- the containments are all proper;
- we know some specific languages that made each containment proper.
- but we have not proved $Hilber^{10}$ is undecidable.

There are two techniques to prove a language $L$ is not decidable

1. directly prove that $L$ is not decidable;
2. use the fact that another language $B$ is not decidable,
   transform the recognition of $B$ to recognition of $L$, i.e., $B \leq L$, 
4.1 Decidable Languages

The relationship among classes of languages

regular ⊂ context-free ⊂ decidable ⊂ Turing-recognizable

• the containments are all proper;
• we know some specific languages that made each containment proper.
• but we have not proved $\text{Hilber10}$ is undecidable.

There are two techniques to prove a language $L$ is not decidable

1. directly prove that $L$ is not decidable;

2. use the fact that another language $B$ is not decidable, transform the recognition of $B$ to recognition of $L$, i.e., $B \leq L$, a decider for $L +$ transformation would yield a decider for $B$, contradict.
Chapter 4. Decidability

4.2 Undecidable languages

We will discuss the following:

1. Cantor's Diagonalization method (first used to prove real numbers are not countable)

2. Show the following language is not decidable

$$A_{TM} = \{\langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \}$$

3. Show the $\text{HALT}_{TM}$ is not decidable, where

$$\text{HALT}_{TM} = \{\langle M, w \rangle : \text{Turing machine } M \text{ halts on } w \}$$

using the result of 2.
4.2 Undecidable languages

1. Cantor's Diagonalization method (first used to prove real numbers are not countable)

2. Show the following language is not decidable
\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

3. Show the \[ HALT_{TM} \] is not decidable, where \[ HALT_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ halts on } w \} \] using the result of 2.
4.2 Undecidable languages

We will discuss the following:

1. Cantor's Diagonalization method (first used to prove real numbers are not countable)
2. Show the following language is not decidable:
   \[ A_{\text{TM}} = \{\langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]
3. Show the \( \text{HALT}_{\text{TM}} \) is not decidable, where
   \[ \text{HALT}_{\text{TM}} = \{\langle M, w \rangle : \text{Turing machine } M \text{ halts on } w \} \]
   using the result of 2.
Chapter 4. Decidability

4.2 Undecidable languages

We will discuss the following:

1. Cantor’s Diagonalization method
   (first used to prove real numbers are not countable)
4.2 Undecidable languages

We will discuss the following:

1. Cantor’s Diagonalization method
   (first used to prove real numbers are not countable)

2. Show the following language is not decidable

   \[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

   using the diagonalization method.
4.2 Undecidable languages

We will discuss the following:

1. Cantor’s Diagonalization method
   (first used to prove real numbers are not countable)

2. Show the following language is not decidable

   \[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

   using the diagonalization method.

3. Show the \( \text{HALT}_{TM} \) is not decidable, where

   \[ \text{HALT}_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ halts on } w \} \]

   using the result of 2.
4.2 Undecidable languages
4.2 Undecidable languages

Diagonalization method (Cantor)
4.2 Undecidable languages

**Diagonalization method** (Cantor)

A set $S$ is **countable** if either

1. $S$ is finite, or
2. there is a correspondence mapping $\phi : S \rightarrow \mathcal{N}$
   where $\mathcal{N}$ is the natural number set.
4.2 Undecidable languages

**Diagonalization method** *(Cantor)*

A set $S$ is **countable** if either

1. $S$ is finite, or
2. there is a correspondence mapping $\phi : S \rightarrow \mathcal{N}$
   where $\mathcal{N}$ is the natural number set.

E.g., $\mathcal{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots \}$, the set of all integers, is countable.
4.2 Undecidable languages

**Diagonalization method** (Cantor)

A set $S$ is **countable** if either

1. $S$ is finite, or
2. there is a correspondence mapping $\phi : S \rightarrow \mathbb{N}$

where $\mathbb{N}$ is the natural number set.

e.g., $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots \}$, the set of all integers, is countable.
e.g., $\Sigma^0$, the set of all binary strings, is countable
4.2 Undecidable languages

**Diagonalization method** (Cantor)

A set $S$ is **countable** if either

1. $S$ is finite, or
2. there is a correspondence mapping $\phi : S \rightarrow \mathbb{N}$ where $\mathbb{N}$ is the natural number set.

E.g., $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots \}$, the set of all integers, is countable.

E.g., $\Sigma^0$, the set of all binary strings, is countable.

E.g., $Q = \{ \frac{m}{n} : m, n \in \mathbb{N} \}$, the set of all rational numbers, is countable.
4.2 Undecidable languages
4.2 Undecidable languages

One-to-one correspondence between \( Q \) and \( N \)

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1/1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & \ldots \\
2 & 2/1 & 2/2 & 2/3 & 2/4 & 2/5 & 2/6 & 2/7 & 2/8 & \ldots \\
4 & 4/1 & 4/2 & 4/3 & 4/4 & 4/5 & 4/6 & 4/7 & 4/8 & \ldots \\
5 & 5/1 & 5/2 & 5/3 & 5/4 & 5/5 & 5/6 & 5/7 & 5/8 & \ldots \\
7 & 7/1 & 7/2 & 7/3 & 7/4 & 7/5 & 7/6 & 7/7 & 7/8 & \ldots \\
8 & 8/1 & 8/2 & 8/3 & 8/4 & 8/5 & 8/6 & 8/7 & 8/8 & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & .
\end{array}
\]

\( N: \) \quad 1, 2, 3, 4, 5, 6, \ldots \\
\( Q: \) \quad 1, 2, \frac{1}{2}, \frac{1}{3}, 3, 4, \ldots
4.2 Undecidable languages

The set $\mathbb{R}$ of real numbers is NOT countable. Proof by contradiction.

Assume that $\mathbb{R}$ is countable. Then there is a one-to-one correspondence between $\mathbb{N}$ and $\mathbb{R}$. Let real number $x = 0.x_1x_2x_3...x_i...$ such that $x_i$ is different from the $i$th digit (after the decimal point) in the $i$th real number. $x$ is NOT in $\mathbb{R}$. (Why?) Contradict!
4.2 Undecidable languages

The set \( \mathbb{R} \) of real numbers is NOT countable.
4.2 Undecidable languages

The set $\mathcal{R}$ of real numbers is NOT countable.

Proof by contradiction.
4.2 Undecidable languages

The set $\mathcal{R}$ of real numbers is NOT countable.

Proof by contradiction. Assume that $\mathcal{R}$ is countable. Then there is a one-to-one correspondence between $\mathcal{N}$ and $\mathcal{R}$.

\[
\begin{align*}
1 & \leftrightarrow 2.397204817\ldots \\
2 & \leftrightarrow 14.526613809\ldots \\
3 & \leftrightarrow 0.498310123\ldots \\
4 & \leftrightarrow 292.275418831\ldots \\
5 & \leftrightarrow 12.002200025\ldots \\
6 & \leftrightarrow 1.999904681\ldots \\
& \vdots \\
0.6 & \leftrightarrow 3175\ldots
\end{align*}
\]
The set $\mathcal{R}$ of real numbers is NOT countable.

Proof by contradiction. Assume that $\mathcal{R}$ is countable. Then there is a one-to-one correspondence between $\mathcal{N}$ and $\mathcal{R}$.

Let real number $x = 0.x_1x_2x_3 \ldots x_i \ldots$
4.2 Undecidable languages

The set $\mathcal{R}$ of real numbers is NOT countable.

Proof by contradiction. Assume that $\mathcal{R}$ is countable. Then there is a one-to-one correspondence between $\mathcal{N}$ and $\mathcal{R}$.

Let real number $x = 0.x_1x_2x_3\ldots x_i\ldots$ such that $x_i$ is different from the $i$th digit (after the decimal point) in the $i$th real number.
4.2 Undecidable languages

The set $\mathcal{R}$ of real numbers is NOT countable.

Proof by contradiction. Assume that $\mathcal{R}$ is countable. Then there is a one-to-one correspondence between $\mathbb{N}$ and $\mathcal{R}$.

Let real number $x = 0.x_1x_2x_3 \ldots x_i \ldots$ such that $x_i$ is different from the $i$th digit (after the decimal point) in the $i$th real number.

$x$ is NOT in $\mathcal{R}$. (Why?)
4.2 Undecidable languages

The set $\mathcal{R}$ of real numbers is NOT countable.

Proof by contradiction. Assume that $\mathcal{R}$ is countable. Then there is a one-to-one correspondence between $\mathbb{N}$ and $\mathcal{R}$.

Let real number $x = 0.x_1 x_2 x_3 \ldots x_i \ldots$ such that $x_i$ is different from the $i$th digit (after the decimal point) in the $i$th real number.

$x$ is NOT in $\mathcal{R}$. (Why?)
Contradict!
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $\text{ALLTM}$:

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $M_0$ | $T$ | $F$ | $F$ | $F$ | $F$ | $F$ | $T$ |
| $M_1$ | $F$ | $F$ | $T$ | $F$ | $T$ | $F$ | $F$ |
| $M_{00}$ | $F$ | $F$ | $F$ | $T$ | $F$ | $F$ | $F$ |
| $M_{01}$ | $T$ | $T$ | $T$ | $T$ | $F$ | $T$ | $T$ |
| $M_{10}$ | $T$ | $F$ | $T$ | $F$ | $T$ | $T$ | $T$ |
| $M_{11}$ | $T$ | $F$ | $F$ | $F$ | $F$ | $F$ | $T$ |
| ... | ... | ... | ... | ... | ... | ... | ... |

where entry $T(M_i, w_j)$ tells TM $M_i$ accepts/rejects input $w_j$.

e.g., $T(M_{00}, 10) = F$ indicates $M_{00}$ rejects string 10.
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;

<table>
<thead>
<tr>
<th>$M$</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>$0$</td>
</tr>
<tr>
<td>$M_{00}$</td>
<td>$1$</td>
</tr>
<tr>
<td>$M_{01}$</td>
<td>$1$</td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>$1$</td>
</tr>
<tr>
<td>$M_{11}$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

$$
\begin{array}{ccccccc}
\epsilon & 0 & 1 & 00 & 01 & 10 & 11 & \ldots & w & \ldots \\
\end{array}
$$
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

<table>
<thead>
<tr>
<th>$M_\epsilon$</th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>$\ldots$</th>
<th>$w$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

<table>
<thead>
<tr>
<th>$w$</th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>...</th>
<th>$M_\epsilon$</th>
<th>$M_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>$0$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$00$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$01$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$11$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$w$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $\textit{ALLTM}$:

\[
\begin{array}{cccccccc}
\epsilon & 0 & 1 & 00 & 01 & 10 & 11 & \ldots \ w \ldots \\
M_\epsilon & T & T & F & T & T & F & F \\
M_0 & T & F & F & F & F & F & T \\
M_1 & F & F & T & F & T & F & F \\
\end{array}
\]
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

|    | $\epsilon$ | 0 | 1 | 00 | 01 | 10 | 11 | ... | $w$ | ...
|----|-------------|---|---|----|----|----|----|-----|-----|-----
| $M_\epsilon$ | T | T | F | T | T | F | F |     |     |     
| $M_0$     | T | F | F | F | F | F | F | T   |     |     
| $M_1$     | F | F | T | F | T | F | F |     |     |     
| $M_{00}$  | F | F | F | T | F | F | F |     |     |     

4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>$\ldots$</th>
<th>$w$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_\epsilon$</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_0$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_1$</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{00}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{01}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>...</th>
<th>$w$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_\epsilon$</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_0$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_1$</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{00}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{01}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example, $T(M_{00}, 10) = F$ indicates $M_{00}$ rejects string $10$. 
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $ALLTM$:

|       | $\epsilon$ | 0 | 1 | 00 | 01 | 10 | 11 | ... | $w$ | ...
|-------|------------|---|---|----|----|----|----|-----|-----|-------
| $M_\epsilon$ | T | T | F | F | T | F | F |     |     |       |
| $M_0$   | T | F | F | F | F | F | F | T |     |       |
| $M_1$   | F | F | T | F | T | F | F |     |       |       |
| $M_{00}$| F | F | F | T | F | F | F | F |     |       |
| $M_{01}$| T | T | T | T | F | T | T |     |       |       |
| $M_{10}$| T | F | F | T | T | T | T |     |       |       |
| $M_{11}$| T | F | F | F | F | F | T |     |       |       |
# 4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $\text{ALLTM}$:

|     | $\epsilon$ | 0   | 1   | 00  | 01  | 10  | 11  | ... | $w$ | ...
|-----|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----
| $M_\epsilon$ | T   | T   | F   | T   | T   | F   | F   |     |     |     
| $M_0$    | T   | F   | F   | F   | F   | F   | T   |     |     |     
| $M_1$    | F   | F   | T   | F   | T   | F   | F   |     |     |     
| $M_{00}$ | F   | F   | F   | T   | F   | F   | F   |     |     |     
| $M_{01}$ | T   | T   | T   | T   | F   | T   | T   |     |     |     
| $M_{10}$ | T   | F   | T   | F   | T   | T   | T   |     |     |     
| $M_{11}$ | T   | F   | F   | F   | F   | F   | T   |     |     |     
| ...     | ...  | ... | ... | ... | ... | ... | ... |     |     |     

For example, $T(M_{00}, 10) = F$ indicates $M_{00}$ rejects string $10$. 
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $\text{ALLTM}$:

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_\epsilon$</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$M_0$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$M_1$</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$M_{00}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$M_{01}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$M_{11}$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_w$</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>
4.2 Undecidable languages

Recall $\Sigma^*$ is countable; therefore,

- the set of all binary strings can be encoded by natural numbers;
- the set of all TMs can be encoded by natural numbers;

There is a table $\text{ALLTM}$:

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>$\ldots$</th>
<th>$w$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_\epsilon$</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_0$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_{00}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_{01}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_{11}$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$\ldots$</td>
<td>$w$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_w$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$T$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

where entry $T(M_i, w_j)$ tells TM $M_i$ accepts/rejects input $w_j$.

e.g., $T(M_{00}, 10) = F$ indicates $M_{00}$ rejects string 10.
4.2 Undecidable languages

Consider language $A_{TM} = \{\langle M, w \rangle : \text{Turing machine } M \text{ accepts } w\}$

Theorem 4.11 $A_{TM}$ is not decidable.

Proof idea:
• Assume otherwise there is a decider $U$ for language $A_{TM}$.
  • We can construct a decider $M_D$ that on input $w$, $M_D$ calls $U$ to decide on $\langle M_w, w \rangle$ and accepts $w$ iff $U$ rejects $\langle M_w, w \rangle$. i.e., $M_D$ does the exact opposite of $U$.
  • Now question: does $M_D$ accept input string $D$?
    If $M_D$ accepts string $D$, then $U$ rejects $\langle M_D, D \rangle$, $M_D$ rejects $D$;
    If $M_D$ rejects string $D$, then $U$ accepts $\langle M_D, D \rangle$, $M_D$ accepts $D$;
  Paradox!
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.

**Proof idea:**
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

...
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

Theorem 4.11 \( A_{TM} \) is not decidable.

Proof idea:
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

• We can construct a decider \( M_D \) that
  On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)


4.2 Undecidable languages

Consider language

$$A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \}$$

**Theorem 4.11** $A_{TM}$ is not decidable.

**Proof idea:** Assume otherwise there is a decider $U$ for language $A_{TM}$.

- We can construct a decider $M_D$ that
  - On input $w$, $M_D$ calls $U$ to decide on $\langle M_w, w \rangle$
  - and accepts $w$ iff $U$ rejects $\langle M_w, w \rangle$
Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.

**Proof idea:**
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

- We can construct a decider \( M_D \) that
  On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)
  and **accepts** \( w \) iff \( U \) **rejects** \( \langle M_w, w \rangle \)

  i.e., \( M_D \) does the exact opposite of \( U \).
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.

**Proof idea:**
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

- We can construct a decider \( M_D \) that
  On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \) and **accepts** \( w \) iff \( U \) **rejects** \( \langle M_w, w \rangle \)
  i.e., \( M_D \) does the exact opposite of \( U \).

- **Now question:** does \( M_D \) **accepts** input string \( D \)?
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{ Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.

**Proof idea:**

Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

- We can construct a decider \( M_D \) that
  - On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)
  - and accepts \( w \) iff \( U \) rejects \( \langle M_w, w \rangle \)

  i.e., \( M_D \) does the exact opposite of \( U \).

- **Now question:** does \( M_D \) accepts input string \( D \)?

  If \( M_D \) accepts string \( D \),
4.2 Undecidable languages

Consider language

$$A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \}$$

Theorem 4.11 \( A_{TM} \) is not decidable.

Proof idea:
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

- We can construct a decider \( M_D \) that
  
  On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)
  and accepts \( w \) iff \( U \) rejects \( \langle M_w, w \rangle \)

  i.e., \( M_D \) does the exact opposite of \( U \).

- Now question: does \( M_D \) accepts input string \( D \) ?
  
  If \( M_D \) accepts string \( D \), then \( U \) rejects \( \langle M_D, D \rangle \),
4.2 Undecidable languages

Consider language

$$A_{TM} = \{\langle M, w \rangle : \text{Turing machine } M \text{ accepts } w\}$$

**Theorem 4.11** $A_{TM}$ is not decidable.

**Proof idea:**
Assume otherwise there is a decider $U$ for language $A_{TM}$.

- We can construct a decider $M_D$ that
  - On input $w$, $M_D$ calls $U$ to decides on $\langle M_w, w \rangle$
  - and **accepts** $w$ iff $U$ **rejects** $\langle M_w, w \rangle$

  i.e., $M_D$ does the exact opposite of $U$.

- **Now question:** does $M_D$ accepts input string $D$ ?

  If $M_D$ accepts string $D$, then $U$ rejects $\langle M_D, D \rangle$, $M_D$ rejects $D$;
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.

**Proof idea:**
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

- We can construct a decider \( M_D \) that
  On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)
  and accepts \( w \) iff \( U \) rejects \( \langle M_w, w \rangle \)

  i.e., \( M_D \) does the exact opposite of \( U \).

- **Now question:** does \( M_D \) accepts input string \( D \) ?
  
  If \( M_D \) accepts string \( D \), then \( U \) rejects \( \langle M_D, D \rangle \), \( M_D \) rejects \( D \);
  
  If \( M_D \) rejects string \( D \),
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

**Theorem 4.11** \( A_{TM} \) is not decidable.

**Proof idea:**
Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

- We can construct a decider \( M_D \) that
  - On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)
  - and accepts \( w \) iff \( U \) rejects \( \langle M_w, w \rangle \)
  
  i.e., \( M_D \) does the exact opposite of \( U \).

- **Now question:** does \( M_D \) accepts input string \( D \)?
  
  If \( M_D \) accepts string \( D \), then \( U \) rejects \( \langle M_D, D \rangle \), \( M_D \) rejects \( D \);
  
  If \( M_D \) rejects string \( D \), then \( U \) accepts \( \langle M_D, D \rangle \),
4.2 Undecidable languages

Consider language

\[ A_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ accepts } w \} \]

Theorem 4.11 \( A_{TM} \) is not decidable.

Proof idea:

Assume otherwise there is a decider \( U \) for language \( A_{TM} \).

• We can construct a decider \( M_D \) that
  On input \( w \), \( M_D \) calls \( U \) to decides on \( \langle M_w, w \rangle \)
  and accepts \( w \) iff \( U \) rejects \( \langle M_w, w \rangle \)

  i.e., \( M_D \) does the exact opposite of \( U \).

• Now question: does \( M_D \) accepts input string \( D \) ?

  If \( M_D \) accepts string \( D \), then \( U \) rejects \( \langle M_D, D \rangle \), \( M_D \) rejects \( D \);
  If \( M_D \) rejects string \( D \), then \( U \) accepts \( \langle M_D, D \rangle \), \( M_D \) accepts \( D \);

  Paradox!
4.2 Undecidable languages

We explain this proof from diagonalization of table $\textsc{ALLTM}$:

\begin{array}{cccccc}
\epsilon & 0 & 1 & 00 & 01 & 10 & 11 \\
\hline
M_\epsilon & & & & & & \\
M_0 & T & F & F & F & F & T \\
M_1 & F & F & T & F & T & F \\
M_{00} & F & F & F & T & F & F \\
M_{01} & T & T & T & T & F & T \\
M_{10} & T & F & T & F & T & T \\
M_{11} & T & F & F & F & F & T \\
\hline
\end{array}

But $M_D$ does not exist in the table because:

- If $U$ accepts $\langle M_D, D \rangle$, it means $M_D$ accepts $D$, but $M_D$ rejects $D$;
- If $U$ rejects $\langle M_D, D \rangle$, it means $M_D$ rejects $D$, but $M_D$ accepts $D$. 
### 4.2 Undecidable languages

We explain this proof from diagonalization of table $ALLTM$:

|     | $\epsilon$ | 0 | 1 | 00 | 01 | 10 | 11 | ... | $D$ | ...
|-----|-------------|---|---|----|----|----|----|-----|-----|-----
| $M_\epsilon$ | T | T | F | F | T | T | F | F |
| $M_0$    | T | F | F | F | F | F | F | T |
| $M_1$    | F | F | T | F | T | F | F |
| $M_{00}$ | F | F | F | T | F | F | F |
| $M_{01}$ | T | T | T | T | F | T | T |
| $M_{10}$ | T | F | T | F | T | T | T |
| $M_{11}$ | T | F | F | F | F | F | T |
| ...     | ...         | ... | ... | ... | ... | ... | ... | ... | ... | ...

$M_D$ does not exist in the table because:

- If $U$ accepts $\langle M_D, D \rangle$, it means $M_D$ accepts $D$, but $M_D$ rejects $D$;
- If $U$ rejects $\langle M_D, D \rangle$, it means $M_D$ rejects $D$, but $M_D$ accepts $D$. 

But $M_D$ does not exist in the table because:
4.2 Undecidable languages

We explain this proof from diagonalization of table $ALLTM$:

|   | $\epsilon$ | 0   | 1   | 00  | 01  | 10  | 11  | ... | $D$ | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_\epsilon$</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_0$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_1$</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{00}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{01}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_{11}$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_D$</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But $M_D$ does not exist in the table because:

- If $U$ accepts $\langle M_D, D \rangle$, it means $M_D$ accepts $D$,
  but $M_D$ rejects $D$;
4.2 Undecidable languages

We explain this proof from diagonalization of table \( ALLTM \):

<table>
<thead>
<tr>
<th></th>
<th>( \epsilon )</th>
<th>0</th>
<th>1</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>...</th>
<th>( D )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_\epsilon )</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_0 )</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_1 )</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_{00} )</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_{01} )</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_{10} )</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_{11} )</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</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>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M_D )</td>
<td></td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>?</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>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But \( M_D \) does not exist in the table because:

- If \( U \) accepts \( \langle M_D, D \rangle \), it means \( M_D \) accepts \( D \), but \( M_D \) rejects \( D \);

- If \( U \) rejects \( \langle M_D, D \rangle \), it means \( M_D \) rejects \( D \), but \( M_D \) accepts \( D \).
4.2 Undecidable languages

HALT

\[ TM = \{ \langle M, w \rangle : \text{Turing machine } M \text{ halts on input } w \} \]

Theorem 5.1 HALT

\[ TM \text{ is undecidable.} \]

Proof idea:

- Transforming recognition of \( A_{TM} \) to recognition of \( HALT_{TM} \), such that if \( HALT_{TM} \) is decidable, so would \( A_{TM} \) be.
- Such a method is called reduction.
4.2 Undecidable languages

Showing other undecidable languages (using Theorem 4.11)

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ halts on input } w \} \]
4.2 Undecidable languages

Showing other undecidable languages (using Theorem 4.11)

$$\text{HALT}_{TM} = \{\langle M, w \rangle : \text{ Turing machine } M \text{ halts on input } w \}$$

**Theorem 5.1** \text{HALT}_{TM} \text{ is undecidable.} \text{(from section }
4.2 Undecidable languages

Showing other undecidable languages (using Theorem 4.11)

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ halts on input } w \} \]

**Theorem 5.1** \( \text{HALT}_{TM} \) is undecidable. (from section

**Proof idea:**

- Transforming recognition of \( A_{TM} \) to recognition of \( \text{HALT}_{TM} \), such that
4.2 Undecidable languages

Showing other undecidable languages (using Theorem 4.11)

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle : \text{Turing machine } M \text{ halts on input } w \} \]

**Theorem 5.1** \( \text{HALT}_{TM} \) is undecidable. (from section

**Proof idea:**

- **Transforming recognition of** \( A_{TM} \) **to recognition of** \( \text{HALT}_{TM} \), **such that**

if \( \text{HALT}_{TM} \) is decidable, so would \( A_{TM} \) be.
4.2 Undecidable languages

Showing other undecidable languages (using Theorem 4.11)

\[
\text{HALT}_{TM} = \{ \langle M, w \rangle : \text{ Turing machine } M \text{ halts on input } w \}\]

**Theorem 5.1 HALT}_{TM} is undecidable.** (from section

**Proof idea:**

- Transforming recognition of \( A_{TM} \) to recognition of \( \text{HALT}_{TM} \), such that
  
  if \( \text{HALT}_{TM} \) is decidable, so would \( A_{TM} \) be.

- such a method is called **reduction.**
4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{\text{TM}}$. We construct a decision $S$ for $\text{A}_{\text{TM}}$ as follows:

- On input $\langle M, w \rangle$,
  - run $R$ on $\langle M, w \rangle$, reject if $R$ rejects (meaning $M$ does not halt on $w$);
  - if $R$ accepts (meaning $M$ halts on $w$), simulate $M$ on $w$, accept if $M$ accepts; reject if $M$ rejects.
4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{TM}$. 

...
4.2 Undecidable languages

Assume that decider $R$ for \( \text{HALT}_{TM} \).
We construct a decide $S$ for $A_{TM}$ as follows:
4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{TM}$.
We construct a decide $S$ for $A_{TM}$ as follows:

On input $\langle M, w \rangle$,
4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{TM}$.
We construct a decide $S'$ for $A_{TM}$ as follows:

On input $\langle M, w \rangle$,
- run $R$ on $\langle M, w \rangle$, 

4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{TM}$.
We construct a decide $S$ for $\text{A}_{TM}$ as follows:

On input $\langle M, w \rangle$,
- run $R$ on $\langle M, w \rangle$,
  * rejects if $R$ rejects (meaning $M$ does not halts on $w$);
4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{TM}$.
We construct a decide $S$ for $A_{TM}$ as follows:

On input $\langle M, w \rangle$,

- run $R$ on $\langle M, w \rangle$,
  - rejects if $R$ rejects (meaning $M$ does not halts on $w$);
- if $R$ accepts (meaning $M$ halts on $w$),
4.2 Undecidable languages

Assume that decider $R$ for $\text{HALT}_{TM}$.
We construct a decide $S$ for $A_{TM}$ as follows:

On input $\langle M, w \rangle$,
- run $R$ on $\langle M, w \rangle$,
  * rejects if $R$ rejects (meaning $M$ does not halts on $w$);
- if $R$ accepts (meaning $M$ halts on $w$),
  * simulates $M$ on $w$, accepts if $M$ accepts; rejects if $M$ rejects.
4.2 Undecidable languages

Another undecidable language: $E_{TM} = \{ \langle M \rangle : M \text{ is Turing machine and } L(M) = \emptyset \}$

**Theorem 5.2** $E_{TM}$ is undecidable.

**Proof idea:** Transforming recognition of $A_{TM}$ to recognition of $E_{TM}$, such that if $E_{TM}$ is decidable, so would $A_{TM}$ be. i.e., information deciding an instance for $E_{TM}$ can be used to decide the instance $\langle M, w \rangle$ for $A_{TM}$.

Given $\langle M, w \rangle$, construct TM $M_1$ of following behavior:

- on input $x$, if $x \neq w$, rejects;
- if $x = w$, then simulates $M$ on $w$, accepts if $M$ does.
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \text{ } M \text{ is Turing machine and } L(M) = \emptyset \} \]
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \text{ } M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.

**Proof idea:**

Transforming recognition of \( A_{TM} \) to recognition of \( E_{TM} \), such that
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \text{ } M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.

**Proof idea:**

Transforming recognition of \( A_{TM} \) to recognition of \( E_{TM} \), such that
if \( E_{TM} \) is decidable, so would \( A_{TM} \) be.
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \text{ } M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.

**Proof idea:**

Transforming recognition of \( A_{TM} \) to recognition of \( E_{TM} \), such that

if \( E_{TM} \) is decidable, so would \( A_{TM} \) be.

i.e., information deciding an instance for \( E_{TM} \) can be used to decide the instance \( \langle M, w \rangle \) for \( A_{TM} \).
Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \text{ } M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.

**Proof idea:**

Transforming recognition of \( A_{TM} \) to recognition of \( E_{TM} \), such that

if \( E_{TM} \) is decidable, so would \( A_{TM} \) be.

i.e., information deciding an instance for \( E_{TM} \) can be used
to decide the instance \( \langle M, w \rangle \) for \( A_{TM} \).

Given \( \langle M, w \rangle \), construct TM \( M_1 \) of following behavior:
on input \( x \),
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \ M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.

**Proof idea:**

Transforming recognition of \( A_{TM} \) to recognition of \( E_{TM} \), such that

if \( E_{TM} \) is decidable, so would \( A_{TM} \) be.

i.e., information deciding an instance for \( E_{TM} \) can be used to decide the instance \( \langle M, w \rangle \) for \( A_{TM} \).

Given \( \langle M, w \rangle \), construct TM \( M_1 \) of following behavior:

on input \( x \),

if \( x \neq w \), rejects;

if \( x = w \), runs \( M \) and accepts if \( L(M) = \emptyset \).
4.2 Undecidable languages

Another undecidable language:

\[ E_{TM} = \{ \langle M \rangle : \ M \text{ is Turing machine and } L(M) = \emptyset \} \]

**Theorem 5.2** \( E_{TM} \) is undecidable.

**Proof idea:**

Transforming recognition of \( A_{TM} \) to recognition of \( E_{TM} \), such that

if \( E_{TM} \) is decidable, so would \( A_{TM} \) be.

i.e., information deciding an instance for \( E_{TM} \) can be used to decide the instance \( \langle M, w \rangle \) for \( A_{TM} \).

Given \( \langle M, w \rangle \), construct TM \( M_1 \) of following behavior:

on input \( x \),

if \( x \neq w \), rejects;

if \( x = w \), then simulates \( M \) on \( w \), accepts if \( M \) does.
4.2 Undecidable languages

Proof:
Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:

On input $\langle M, w \rangle$;

• produces $M_1$ as previously described;

• runs $R$ on $\langle M_1 \rangle$;

rejects if $R$ accepts (meaning $L(M_1) = \emptyset$, $M$ does not accept $w$),

accepts if $R$ rejects (meaning $L(M_1) \neq \emptyset$, $M$ accepts $w$).
4.2 Undecidable languages

Proof:
Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$. 
4.2 Undecidable languages

Proof:

Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:
4.2 Undecidable languages

Proof:

Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:

On input $\langle M, w \rangle$;
4.2 Undecidable languages

Proof:
Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:

On input $\langle M, w \rangle$;

• produces $M_1$ as previously described;
4.2 Undecidable languages

Proof:

Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:

On input $\langle M, w \rangle$;

- produces $M_1$ as previously described;
- runs $R$ on $\langle M_1 \rangle$;
4.2 Undecidable languages

Proof:

Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:

On input $\langle M, w \rangle$;
1. produces $M_1$ as previously described;
2. runs $R$ on $\langle M_1 \rangle$;
3. rejects if $R$ accepts (meaning $L(M_1) = \emptyset$, $M$ does not accept $w$),
4.2 Undecidable languages

Proof:
Assume that $E_{TM}$ is decidable. Then there is a decider $R$ for $E_{TM}$.

Construct decider $S$ as follows for $A_{TM}$:

On input $\langle M, w \rangle$;

- produces $M_1$ as previously described;
- runs $R$ on $\langle M_1 \rangle$;
  
  rejects if $R$ accepts (meaning $L(M_1) = \emptyset$, $M$ does not accept $w$),
  
  accepts if $R$ rejects (meaning $L(M_1) \neq \emptyset$, $M$ accepts $w$)
4.2 Undecidable languages
4.2 Undecidable languages

Corollary: The following languages are not Turing recognizable:

\( A_{TM}, \overline{\text{HALT}}_{TM}, \overline{E_{TM}} \)
Chapter 5. Reducibility

Mapping Reduction: A technique to transform one language \( A \) to another language \( B \), denoted as \( A \leq_m B \), is a mapping from \( \Sigma^* \) to \( \Sigma^* \) such that

- positive instances of \( A \) are mapped to positive instances of \( B \)
- negative instances of \( A \) are mapped to negative instances of \( B \)

So the answers to language \( B \) can be used for answers to language \( A \).
Chapter 5. Reducibility

Mapping Reduction: A technique to transform one language $A$ to another language $B$, denoted as $A \leq_m B$, is a mapping from $\Sigma^*$ to $\Sigma^*$ such that:

- positive instances of $A$ are mapped to positive instances of $B$,
- negative instances of $A$ are mapped to negative instances of $B$.

So the answers to language $B$ can be used for answers to language $A$. 
Chapter 5. Reducibility

Mapping Reduction: A technique to transform one language $A$ to another language $B$, denoted as $A \leq_m B$, is a mapping from $\Sigma^*$ to $\Sigma^*$ such that

- positive instances of $A$ are mapped to positive instances of $B$
- negative instances of $A$ are mapped to negative instances of $B$
Chapter 5. Reducibility

**Mapping Reduction:** A technique to transform one language $A$ to another language $B$, denoted as $A \leq_m B$, is a mapping from $\Sigma^*$ to $\Sigma^*$ such that

- **positive instances** of $A$ are mapped to **positive instances** of $B$
- **negative instances** of $A$ are mapped to **negative instances** of $B$
Chapter 5. Reducibility

Mapping Reduction: A technique to transform one language $A$ to another language $B$, denoted as $A \leq_m B$, is a mapping from $\Sigma^*$ to $\Sigma^*$ such that

- positive instances of $A$ are mapped to positive instances of $B$
- negative instances of $A$ are mapped to negative instances of $B$

So the answers to language $B$ can be used for answers to language $A$. 

Chapter 5. Reducibility

The consequence of reduction $A \leq_m B$ is that:

1. If $B$ is decidable, so is $A$;
2. If $A$ is not decidable, neither is $B$;
3. If $B$ is Turing recognizable, so is $A$;
4. If $A$ is not Turing recognizable, neither is $B$.
The consequence of reduction $A \leq_m B$ is that
Chapter 5. Reducibility

The consequence of reduction $A \leq_m B$ is that

(1) If $B$ is decidable, so is $A$;
Chapter 5. Reducibility

The consequence of reduction $A \leq_m B$ is that

(1) If $B$ is decidable, so is $A$;

(2) If $A$ is not decidable, neither is $B$. 
The consequence of reduction $A \leq_m B$ is that

(1) If $B$ is decidable, so is $A$;

(2) If $A$ is not decidable, neither is $B$.

(3) If $B$ is Turing recognizable, so is $A$;
Chapter 5. Reducibility

The consequence of reduction $A \leq_m B$ is that

1. If $B$ is decidable, so is $A$;
2. If $A$ is not decidable, neither is $B$.
3. If $B$ is Turing recognizable, so is $A$;
4. If $A$ is not Turing recognizable, neither is $B$. 
Chapter 5. Reducibility

Example of $\leq_m$.

Define $\text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \}$

Theorem 5.4 $\text{EQ}_{\text{TM}}$ is undecidable.

Proof idea: we construct a mapping reduction $E_{\text{TM}} \leq_m \text{EQ}_{\text{TM}}$

Recall: $E_{\text{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \}$ and we know that $E_{\text{TM}}$ is undecidable (Theorem 5.2).
Example of $\leq_m$.weis
Example of $\leq_m$.

Define

$$EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$
Example of \( \leq_m \).

Define

\[
EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \}
\]

**Theorem 5.4** \( EQ_{TM} \) is undecidable.
Chapter 5. Reducibility

Example of \( \leq_m \).

Define

\[
EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \}
\]

**Theorem 5.4** \( EQ_{TM} \) is undecidable.

Proof idea: we construct a mapping reduction \( ETM \leq_m EQ_{TM} \)
Chapter 5. Reducibility

Example of $\leq_m$.

Define

$$EQ_{TM} = \{\langle M_1, M_2 \rangle | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

**Theorem 5.4** $EQ_{TM}$ is undecidable.

Proof idea: we construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$

Recall:

$$E_{TM} = \{\langle M \rangle | M \text{ is a TM and } L(M) = \emptyset\}$$
Chapter 5. Reducibility

Example of $\leq_m$.

Define

$$EQ_{TM} = \{\langle M_1, M_2 \rangle | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

**Theorem 5.4** $EQ_{TM}$ is undecidable.

Proof idea: we construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$

Recall:

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

and we know that $E_{TM}$ is undecidable (Theorem 5.2).
Chapter 5. Reducibility

Theorem 5.4

**EQ** is undecidable.

**Proof idea:**
• Construct a mapping reduction $E_{TM} \leq m EQ_{TM}$ through function $f: \Sigma^* \rightarrow \Sigma^*$ such that $f(\langle M \rangle) = \langle M, M_1 \rangle$ where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$.
• Then if $\langle M \rangle \in E_{TM}$ then $L(M) = \emptyset$, implying $\langle M, M_1 \rangle \in EQ_{TM}$; if $\langle M \rangle \not\in E_{TM}$, then $L(M) \neq \emptyset$, implying $\langle M, M_1 \rangle \not\in EQ_{TM}$.
• That is: (1) $\langle M \rangle \in E_{TM}$ if and only if $\langle M, M_1 \rangle \in EQ_{TM}$.
• (2) $f$ is computable, so $E_{TM} \leq m EQ_{TM}$ is a valid mapping reduction.
• Thus, $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

**Proof idea:**

• Construct a mapping reduction $E_{TM} \leq m EQ_{TM}$ through function $f$: $\Sigma^* \rightarrow \Sigma^*$ such that $f(\langle M \rangle) = \langle M, M_1 \rangle$ where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$.

• Then if $\langle M \rangle \in E_{TM}$ then $L(M) = \emptyset$, implying $\langle M, M_1 \rangle \in EQ_{TM}$; if $\langle M \rangle \not\in E_{TM}$, then $L(M) \neq \emptyset$, implying $\langle M, M_1 \rangle \not\in EQ_{TM}$.

• That is: (1) $\langle M \rangle \in E_{TM}$ if and only if $\langle M, M_1 \rangle \in EQ_{TM}$.

• $f$ is computable, so $E_{TM} \leq m EQ_{TM}$ is a valid mapping reduction.

• Thus, $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Chapter 5. Reducibility

Theorem 5.4 $EQ_{TM}$ is undecidable.
Proof idea:

• Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$

This construction implies that $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

**Proof idea:**

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f : \Sigma^* \rightarrow \Sigma^*$ such that

  \[ f(\langle M \rangle) = \]

- Thus, $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

**Proof idea:**

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f : \Sigma^* \to \Sigma^*$ such that
  
  $$f(\langle M \rangle) = \langle M, M_1 \rangle$$

- That is: (1) $\langle M \rangle \in E_{TM}$ if and only if $\langle M, M_1 \rangle \in EQ_{TM}$.
- (2) $f$ is computable, so $E_{TM} \leq_m EQ_{TM}$ is a valid mapping reduction.
- Thus, $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Theorem 5.4 \( EQ_{TM} \) is undecidable.

Proof idea:

- Construct a mapping reduction \( E_{TM} \leq_m EQ_{TM} \) through function \( f : \Sigma^* \rightarrow \Sigma^* \) such that
  \[
  f(\langle M \rangle) = \langle M, M_1 \rangle
  \]
  where \( M_1 \) is a Turing machine that accepts no string at all,
Theorem 5.4 $EQ_{TM}$ is undecidable.

Proof idea:

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$f(\langle M \rangle) = \langle M, M_1 \rangle$$

where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$. 

Thus, $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

**Proof idea:**

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f : \Sigma^* \rightarrow \Sigma^*$ such that
  \[
  f(\langle M \rangle) = \langle M, M_1 \rangle
  \]
  where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$.

- Then
  if $\langle M \rangle \in E_{TM}$ then $L(M) = \emptyset$, implying $\langle M, M_1 \rangle \in EQ_{TM}$;
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

*Proof idea:*

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$f(\langle M \rangle) = \langle M, M_1 \rangle$$

where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$.

- Then
  - if $\langle M \rangle \in E_{TM}$ then $L(M) = \emptyset$, implying $\langle M, M_1 \rangle \in EQ_{TM}$;
  - if $\langle M \rangle \notin E_{TM}$, then $L(M) \neq \emptyset$, implying $\langle M, M_1 \rangle \notin EQ_{TM}$. 
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

Proof idea:

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f : \Sigma^* \rightarrow \Sigma^*$ such that
  
  $$f(\langle M \rangle) = \langle M, M_1 \rangle$$

  where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$.

- Then
  
  if $\langle M \rangle \in E_{TM}$ then $L(M) = \emptyset$, implying $\langle M, M_1 \rangle \in EQ_{TM}$;
  
  if $\langle M \rangle \not\in E_{TM}$, then $L(M) \neq \emptyset$, implying $\langle M, M_1 \rangle \not\in EQ_{TM}$.

- That is: (1) $\langle M \rangle \in E_{TM}$ if and only if $\langle M, M_1 \rangle \in EQ_{TM}$. 
Chapter 5. Reducibility

**Theorem 5.4** \( EQ_{TM} \) is undecidable.

**Proof idea:**

- Construct a mapping reduction \( E_{TM} \leq_m EQ_{TM} \)
  through function \( f : \Sigma^* \to \Sigma^* \) such that

\[
f(\langle M \rangle) = \langle M, M_1 \rangle
\]

where \( M_1 \) is a Turing machine that accepts no string at all, i.e., \( L(M_1) = \emptyset \).

- Then
  - if \( \langle M \rangle \in E_{TM} \) then \( L(M) = \emptyset \), implying \( \langle M, M_1 \rangle \in EQ_{TM} \);
  - if \( \langle M \rangle \notin E_{TM} \), then \( L(M) \neq \emptyset \), implying \( \langle M, M_1 \rangle \notin EQ_{TM} \).

- That is: (1) \( \langle M \rangle \in E_{TM} \) if and only if \( \langle M, M_1 \rangle \in EQ_{TM} \).

- (2) \( f \) is computable, so \( E_{TM} \leq_m EQ_{TM} \) is a valid mapping reduction.
Theorem 5.4 \( EQ_{TM} \) is undecidable.

Proof idea:

- Construct a mapping reduction \( E_{TM} \leq_m EQ_{TM} \) through function \( f : \Sigma^* \rightarrow \Sigma^* \) such that
  \[
  f(\langle M \rangle) = \langle M, M_1 \rangle
  \]
  where \( M_1 \) is a Turing machine that accepts no string at all, i.e., \( L(M_1) = \emptyset \).

- Then
  if \( \langle M \rangle \in E_{TM} \) then \( L(M) = \emptyset \), implying \( \langle M, M_1 \rangle \in EQ_{TM} \);
  if \( \langle M \rangle \notin E_{TM} \), then \( L(M) \neq \emptyset \), implying \( \langle M, M_1 \rangle \notin EQ_{TM} \).

- That is: (1) \( \langle M \rangle \in E_{TM} \) if and only if \( \langle M, M_1 \rangle \in EQ_{TM} \).

- (2) \( f \) is computable, so \( E_{TM} \leq_m EQ_{TM} \) is a valid mapping reduction.

- Thus, \( EQ_{TM} \) is undecidable.
Chapter 5. Reducibility

**Theorem 5.4** $EQ_{TM}$ is undecidable.

**Proof idea:**

- Construct a mapping reduction $E_{TM} \leq_m EQ_{TM}$ through function $f: \Sigma^* \rightarrow \Sigma^*$ such that
  
  $$f(\langle M \rangle) = \langle M, M_1 \rangle$$
  
  where $M_1$ is a Turing machine that accepts no string at all, i.e., $L(M_1) = \emptyset$.

- Then
  
  - if $\langle M \rangle \in E_{TM}$ then $L(M) = \emptyset$, implying $\langle M, M_1 \rangle \in EQ_{TM}$;
  - if $\langle M \rangle \not\in E_{TM}$, then $L(M) \neq \emptyset$, implying $\langle M, M_1 \rangle \not\in EQ_{TM}$.

- That is: (1) $\langle M \rangle \in E_{TM}$ if and only if $\langle M, M_1 \rangle \in EQ_{TM}$.

- (2) $f$ is computable, so $E_{TM} \leq_m EQ_{TM}$ is a valid mapping reduction.

- Thus, $EQ_{TM}$ is undecidable because $E_{TM}$ is undecidable.
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)

Study languages that whose recognition may not need the full strength of a Turing machine (called resource-bounded).

- the number of steps is limited - time complexity (CPU)
- the amount of tape (space) is limited - space complexity (Memory)
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)

Study languages that whose recognition may not need the full strength of a Turing machine.
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)

Study languages that whose recognition may not need the full strength of a Turing machine (called resource-bounded).
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)

Study languages that whose recognition may not need the full strength of a Turing machine (called resource-bounded).

For example,
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)

Study languages that whose recognition may not need the full strength of a Turing machine (called resource-bounded).

For example,

- the number of steps is limited - time complexity (CPU)
Part III. Computational Complexity Theory (Chapters 7, 8, 9, 10)

Study languages that whose recognition may not need the full strength of a Turing machine (called resource-bounded).

For example,

- the number of steps is limited - **time complexity** (CPU)
- the amount of tape (space) is limited - **space complexity** (Memory)
Definition: Let $t(n) \geq n$ be an integer function. A $t(n)$-time Turing machine is a Turing machine that moves at most $t(n)$ steps before it halts on inputs encoded by $n$ binary bits.

Recall that Turing machines are your Java programs.

• What Java programs does a $t(n)$-time TM correspond to?

• Why do we require that $t(n) \geq n$?

• Can you give an example of Java program corresponding to a $t(n)$-time TM, for some function $t(n)$?

• Assume $t(n) = n^2$, what can an $n^2$-time Java program do?

• Assume $t(n) = n!$, what can an $n!$-time Java program do?
**Definition:** Let $t(n) \geq n$ be an integer function. A $t(n)$-time *Turing machine* is a Turing machine that moves at most $t(n)$ steps before it halts on inputs encoded by $n$ binary bits.
**Definition:** Let $t(n) \geq n$ be an integer function. A $t(n)$-time Turing machine is a Turing machine that moves at most $t(n)$ steps before it halts on inputs encoded by $n$ binary bits.

Recall that *Turing machines are your Java programs.*
**Definition:** Let \( t(n) \geq n \) be an integer function. A \( t(n) \)-time Turing machine is a Turing machine that moves at most \( t(n) \) steps before it halts on inputs encoded by \( n \) binary bits.

Recall that *Turing machines are your Java programs.*

- What Java programs does a \( t(n) \)-time TM correspond to?

  Assume \( t(n) = n^2 \), what can an \( n^2 \)-time Java program do?

  Assume \( t(n) = n! \), what can an \( n! \)-time Java program do?
**Definition:** Let \( t(n) \geq n \) be an integer function. A \( t(n) \)-time Turing machine is a Turing machine that moves at most \( t(n) \) steps before it halts on inputs encoded by \( n \) binary bits.

Recall that *Turing machines are your Java programs.*

- What Java programs does a \( t(n) \)-time TM correspond to?
- Why do we require that \( t(n) \geq n \)?
Definition: Let $t(n) \geq n$ be an integer function. A $t(n)$-time Turing machine is a Turing machine that moves at most $t(n)$ steps before it halts on inputs encoded by $n$ binary bits.

Recall that Turing machines are your Java programs.

- What Java programs does a $t(n)$-time TM correspond to?
- Why do we require that $t(n) \geq n$?
- Can you give an example of Java program corresponding to a $t(n)$-time TM, for some function $t(n)$?
Definition: Let $t(n) \geq n$ be an integer function. A $t(n)$-time Turing machine is a Turing machine that moves at most $t(n)$ steps before it halts on inputs encoded by $n$ binary bits.

Recall that Turing machines are your Java programs.

- What Java programs does a $t(n)$-time TM correspond to?
- Why do we require that $t(n) \geq n$?
- Can you give an example of Java program corresponding to a $t(n)$-time TM, for some function $t(n)$?
- Assume $t(n) = n^2$, what can an $n^2$-time Java program do?
Definition: Let \( t(n) \geq n \) be an integer function. A \( t(n) \)-time Turing machine is a Turing machine that moves at most \( t(n) \) steps before it halts on inputs encoded by \( n \) binary bits.

Recall that Turing machines are your Java programs.

- What Java programs does a \( t(n) \)-time TM correspond to?
- Why do we require that \( t(n) \geq n \)?
- Can you give an example of Java program corresponding to a \( t(n) \)-time TM, for some function \( t(n) \)?
- Assume \( t(n) = n^2 \), what can an \( n^2 \)-time Java program do?
- Assume \( t(n) = n! \), what can an \( n! \)-time Java program do?
Introduction to Computational Complexity Theory

Definition: Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$\text{TIME}(n^2)$ includes those languages that can be recognized by Java programs that move at most 1 step, 1000 steps, $n$ steps, 600 steps, $5n^2 + 200$ steps, or $790n^2 - 3n + 8$ steps.
Definition: Let $t(n) \geq n$ be an integer function. Then $TIME(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$TIME(n^2)$ includes those languages that can be recognized by Java programs that move at most $1$ step, $1000$ steps, $n$ steps, $600n$ steps, $5n^2 + 200n$ steps, or $790n^2 - 3n + 8$ steps.
Definition: Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$. 
**Definition:** Let \( t(n) \geq n \) be an integer function. Then \( \text{TIME}(t(n)) \) is the class of all languages that can be recognized by \( l(n) \)-time Turing machines, where function \( l(n) \leq ct(n) \) for some constant \( c > 0 \).

\( \text{TIME}(n^2) \) includes those languages that can be recognized by Java programs that move at most
**Definition:** Let \( t(n) \geq n \) be an integer function. Then \( \text{TIME}(t(n)) \) is the class of all languages that can be recognized by \( l(n) \)-time Turing machines, where function \( l(n) \leq ct(n) \) for some constant \( c > 0 \).

\( \text{TIME}(n^2) \) includes those languages that can be recognized by Java programs that move at most 1 step,
**Definition:** Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$\text{TIME}(n^2)$ includes those languages that can be recognized by Java programs that move at most 1 step, 1000 steps,
**Definition:** Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$\text{TIME}(n^2)$ includes those languages that can be recognized by Java programs that move at most
1 step,
1000 steps,
n steps,
**Definition:** Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$\text{TIME}(n^2)$ includes those languages that can be recognized by Java programs that move at most

1 step,
1000 steps,
$n$ steps,
$600n$ steps,
Definition: Let $t(n) \geq n$ be an integer function. Then \( \text{TIME}(t(n)) \) is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

\( \text{TIME}(n^2) \) includes those languages that can be recognized by Java programs that move at most
1 step,
1000 steps,
$n$ steps,
$600n$ steps,
$5n^2 + 200n$ steps,
**Definition:** Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$\text{TIME}(n^2)$ includes those languages that can be recognized by Java programs that move at most
1 step,
1000 steps,
$n$ steps,
$600n$ steps,
$5n^2 + 200n$ steps, or
**Definition:** Let $t(n) \geq n$ be an integer function. Then $\text{TIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

$\text{TIME}(n^2)$ includes those languages that can be recognized by Java programs that move at most
1 step,
1000 steps,
$n$ steps,
$600n$ steps,
$5n^2 + 200n$ steps, or
$790n^2 - 3n + 8$ steps.
Definition: Let $t(n) \geq n$ be an integer function. Then $\text{NTIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time non-deterministic Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

Are time-bounded deterministic and nondeterministic TMs equivalent? Actually, direct simulation of nondeterministic computation by deterministic computation may result in an exponential increase in time cost. So far, we do not know how to do better, unfortunately!
Definition: Let $t(n) \geq n$ be an integer function. Then $\text{NTIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time non-deterministic Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$. Are time-bounded deterministic and nondeterministic TMs equivalent? Actually, direct simulation of nondeterministic computation by deterministic computation may result in an exponential increase in time cost. So far, we do not know how to do better, unfortunately!
**Definition:** Let \( t(n) \geq n \) be an integer function. Then \( \text{NTIME}(t(n)) \) is the class of all languages that can be recognized by \( l(n) \)-time non-deterministic Turing machines, where function \( l(n) \leq ct(n) \) for some constant \( c > 0 \).

Are time-bounded deterministic and nondeterministic TMs equivalent?
**Definition:** Let \( t(n) \geq n \) be an integer function. Then \( \text{NTIME}(t(n)) \) is the class of all languages that can be recognized by \( l(n) \)-time non-deterministic Turing machines, where function \( l(n) \leq ct(n) \) for some constant \( c > 0 \).

Are time-bounded deterministic and nondeterministic TMs equivalent?

Actually, direct simulation of nondeterministic computation by deterministic computation may result in an exponential increase in time cost.
Definition: Let $t(n) \geq n$ be an integer function. Then $\text{NTIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time non-deterministic Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

Are time-bounded deterministic and nondeterministic TMs equivalent?

Actually, direct simulation of nondeterministic computation by deterministic computation may result in an exponential increase in time cost.

So far, we do not know how to do better,
**Definition**: Let $t(n) \geq n$ be an integer function. Then $\text{NTIME}(t(n))$ is the class of all languages that can be recognized by $l(n)$-time non-deterministic Turing machines, where function $l(n) \leq ct(n)$ for some constant $c > 0$.

Are time-bounded deterministic and nondeterministic TMs equivalent?

Actually, direct simulation of nondeterministic computation by deterministic computation may result in an exponential increase in time cost.

So far, we do not know how to do better, unfortunately!
Theorem: Let $f(n) \geq n$ and $g(n) \geq n$ be two functions such that 
\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0. \]

Then the following containment is proper:

\[ \text{TIME}(f(n)) \subset \text{TIME}(g(n)) \]

For example, for any $k > 1$,

\[ \text{TIME}(n) \subset \text{TIME}(n^2) \subset \text{TIME}(n^3) \subset \cdots \subset \text{TIME}(n^k) \]

Where are these classes in the Chomsky Hierarchy?
**Theorem:** Let \( f(n) \geq n \) and \( g(n) \geq n \) be two functions such that 
\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.
\]
Then the following containment is proper:

\[
\text{TIME}(f(n)) \subset \text{TIME}(g(n))
\]
**Theorem:** Let $f(n) \geq n$ and $g(n) \geq n$ be two functions such that \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \). Then the following containment is proper:

\[
\text{TIME}(f(n)) \subset \text{TIME}(g(n))
\]

For example, for any $k > 1$,

\[
\text{TIME}(n) \subset \text{TIME}(n^2) \subset \text{TIME}(n^3) \subset \cdots \subset \text{TIME}(n^k)
\]
**Theorem:** Let \( f(n) \geq n \) and \( g(n) \geq n \) be two functions such that

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.
\]

Then the following containment is proper:

\[
\text{TIME}(f(n)) \subset \text{TIME}(g(n))
\]

For example, for any \( k > 1 \),

\[
\text{TIME}(n) \subset \text{TIME}(n^2) \subset \text{TIME}(n^3) \subset \cdots \subset \text{TIME}(n^k)
\]

Where are these classes in the Chomsky Hierarchy?
Introduction to Computational Complexity Theory

Major tasks in algorithms and complexity theory research:

1. If $L \in \text{TIME}(g(n))$, can we identify $f(n)$, where $\lim_{n \to \infty} f(n) g(n) = 0$, such that $L \in \text{TIME}(f(n))$? That is, can we find strictly faster programs for $L$?

2. If $L \in \text{TIME}(g(n))$, can we prove that for every function $f(n)$, where $\lim_{n \to \infty} f(n) g(n) = 0$, $L \not\in \text{TIME}(f(n))$? That is, can we prove that there are no strictly faster programs for $L$? [lower bound study]
Major tasks in algorithms and complexity theory research:

1. If \( L \in \text{TIME}(g(n)) \), can we identify \( f(n) \), where \( \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \), such that \( L \in \text{TIME}(f(n)) \)? That is, can we find strictly faster programs for \( L \)?

2. If \( L \in \text{TIME}(g(n)) \), can we prove that for every function \( f(n) \), where \( \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \), \( L \not\in \text{TIME}(f(n)) \)? That is, can we prove that there are no strictly faster programs for \( L \)?
Major tasks in algorithms and complexity theory research:

(1) If $L \in \text{TIME}(g(n))$, can we identify $f(n)$, where

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$

such that $L \in \text{TIME}(f(n))$?
Major tasks in algorithms and complexity theory research:

(1) If \( L \in \text{TIME}(g(n)) \), can we identify \( f(n) \), where
\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,
\]
such that \( L \in \text{TIME}(f(n)) \)?

That is, can we find strictly faster programs for \( L \)?

[upper bound study]
Major tasks in algorithms and complexity theory research:

1. If $L \in \text{TIME}(g(n))$, can we identify $f(n)$, where
   \[
   \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,
   \]
   such that $L \in \text{TIME}(f(n))$?
   That is, can we find strictly faster programs for $L$?
   [upper bound study]

2. If $L \in \text{TIME}(g(n))$, can we prove that for every function $f(n)$, where
   \[
   \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0, \quad L \notin \text{TIME}(f(n))?
   \]
Major tasks in algorithms and complexity theory research:

(1) If $L \in \text{TIME}(g(n))$, can we identify $f(n)$, where $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$, such that $L \in \text{TIME}(f(n))$? That is, can we find strictly faster programs for $L$? [upper bound study]

(2) If $L \in \text{TIME}(g(n))$, can we prove that for every function $f(n)$, where $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$, $L \notin \text{TIME}(f(n))$? That is, can we prove that there are no strictly faster programs for $L$? [lower bound study]
Introduction to Computational Complexity Theory

The most well-known open questions:

\[ P = \? \quad \text{NP} \]

where

\[ P = \bigcup_{k=0}^{\infty} \text{TIME}(n^k) \]

\[ \text{NP} = \bigcup_{k=0}^{\infty} \text{NTIME}(n^k) \]

Why is the question important?

Because \( \text{NP} \) contains many problems/languages of practical importance.
Introduction to Computational Complexity Theory

The most well-known open questions:

\[ \text{P} = \text{NP} \]

Where

\[ \text{P} = \bigcup_{k=0}^{\infty} \text{TIME}(n^k) \]

\[ \text{NP} = \bigcup_{k=0}^{\infty} \text{NTIME}(n^k) \]

Why is the question important?

Because \text{NP} contains many problems/languages of practical importance.
The most well-known open questions: $P = ? NP$
The most well-known open questions: \( P =? \ NP \)

where

\[
P = \bigcup_{k=0}^{\infty} \text{TIME}(n^k)
\]
Introduction to Computational Complexity Theory

The most well-known open questions: $P = \? NP$

where

$$P = \bigcup_{k=0}^{\infty} \text{TIME}(n^k) \quad \text{and} \quad NP = \bigcup_{k=0}^{\infty} \text{NTIME}(n^k)$$
The most well-known open questions: \( P =? NP \)

where

\[
P = \bigcup_{k=0}^{\infty} \text{TIME}(n^k) \quad \text{NP} = \bigcup_{k=0}^{\infty} \text{NTIME}(n^k)
\]

Why is the question important? Because \( \text{NP} \) contains many problems/languages of practical importance.
Definition: Let \( s(n) \geq n \) be an integer function. A \( s(n) \)-space Turing machine is a Turing machine that uses at most \( s(n) \) tape cells before it halts on inputs encoded by \( n \) binary bits.

Definition: Let \( s(n) \geq n \) be an integer function. Then \( \text{SPACE}(s(n)) \) is the class of all languages that can be recognized by \( r(n) \)-space Turing machines, where function \( r(n) \leq cs(n) \) for some constant \( c > 0 \).
**Definition:** Let $s(n) \geq n$ be an integer function. A $s(n)$-space Turing machine is a Turing machine that uses at most $s(n)$ tape cells before it halts on inputs encoded by $n$ binary bits.
Definition: Let $s(n) \geq n$ be an integer function. A $s(n)$-space Turing machine is a Turing machine that uses at most $s(n)$ tape cells before it halts on inputs encoded by $n$ binary bits.

Definition: Let $s(n) \geq n$ be an integer function. Then $\text{SPACE}(s(n))$ is the class of all languages that can be recognized by $r(n)$-space Turing machines, where function $r(n) \leq cs(n)$ for some constant $c > 0$. 
Introduction to Computational Complexity Theory

• $\text{TIME}(t(n)) \subseteq \text{SPACE}(t(n))$

• $\text{SPACE}(s(n)) \subseteq \text{TIME}(2^{s'(n)})$, where $s'(n) \geq cs(n)$, for some constant $c > 0$. 
Introduction to Computational Complexity Theory

Relationships between time and space complexities

\[ \text{TIME}(t(n)) \subseteq \text{SPACE}(t(n)) \]

\[ \text{SPACE}(s(n)) \subseteq \text{TIME}(2^{s'(n)}) \]

where \( s'(n) \geq cs(n) \), for some constant \( c > 0 \).
Introduction to Computational Complexity Theory

Relationships between time and space complexities

- $\text{TIME}(t(n)) \subseteq \text{SPACE}(t(n))$
Introduction to Computational Complexity Theory

Relationships between time and space complexities

- $\text{TIME}(t(n)) \subseteq \text{SPACE}(t(n))$
- $\text{SPACE}(s(n)) \subseteq \text{TIME}(2^{s'(n)})$, where $s'(n) \geq cs(n)$, for some constant $c > 0$. 
Topics for the Final Exam

1. Languages and models
   - DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
     - model definitions, definition of accepting a string (and definition of recognizing a language)
   - Chomsky hierarchy:
     - Regular $\subset$ Context free $\subset$ Decidable $\subset$ Turing recognizable $\subset$ All
     - why containments are proper
   - regular language pumping lemma

2. Operations on regular, context-free, Turing decidable, and Turing recognizable languages:
   - union, intersection, complementary, concatenation, star
Topics for the Final Exam

1. Languages and models

   - DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
   - model definitions, definition of accepting a string (and definition of recognizing a language)
   - Chomsky hierarchy: Regular $\subset$ Context free $\subset$ Decidable $\subset$ Turing recognizable $\subset$ All
   - why containments are proper
   - regular language pumping lemma
   - Operations on regular, context-free, Turing decidable, and Turing recognizable languages: union, intersection, complementary, concatenation, star
Topics for the Final Exam

1. Languages and models
   - DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
Topics for the Final Exam

1. Languages and models
   - DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
     - model definitions, definition of accepting a string
       (and definition of recognizing a language)
Topics for the Final Exam

1. Languages and models

   • DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
     - model definitions, definition of accepting a string
       (and definition of recognizing a language)

   • Chomsky hierarchy:
     Regular \( \subset \) Context free \( \subset \) Decidable \( \subset \) Turing recognizable \( \subset \) All
1. Languages and models

- DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
  - model definitions, definition of accepting a string
  - (and definition of recognizing a language)

- Chomsky hierarchy:
  Regular $\subset$ Context free $\subset$ Decidable $\subset$ Turing recognizable $\subset$ All

- why containments are proper
1. Languages and models
   - DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
     - model definitions, definition of accepting a string
     (and definition of recognizing a language)
   - Chomsky hierarchy:
     - Regular $\subset$ Context free $\subset$ Decidable $\subset$ Turing recognizable $\subset$ All
     - why containments are proper
     - regular language pumping lemma
1. Languages and models

- DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
  - model definitions, definition of accepting a string
    (and definition of recognizing a language)

- Chomsky hierarchy:
  Regular \(\subset\) Context free \(\subset\) Decidable \(\subset\) Turing recognizable \(\subset\) All
  - why containments are proper
  - regular language pumping lemma
1. Languages and models

- DFA, NFA, RE, PDA, CFG, Turing-decider, Turing machine
  - model definitions, definition of accepting a string
    (and definition of recognizing a language)

- Chomsky hierarchy:
  Regular $\subset$ Context free $\subset$ Decidable $\subset$ Turing recognizable $\subset$ All
  - why containments are proper
  - regular language pumping lemma

- Operations on regular, context-free, Turing decidable, and Turing recognizable languages:
  union, intersection, complementary, concatenation, star
Topics for the Final Exam

2. Turing machines
   • basics of a TM, definition of TM computation, TM diagram,
   • tracing a TM on a given input
   • designing simple TMs (diagrams)
   • equivalent models of TMs, non-deterministic TM
Topics for the Final Exam

2. Turing machines
2. Turing machines

- basics of a TM, definition of TM computation, TM diagram,
2. Turing machines

- basics of a TM, definition of TM computation, TM diagram,
- tracing a TM on a given input
2. Turing machines

- basics of a TM, definition of TM computation, TM diagram,
- tracing a TM on a given input
- designing simple TMs (diagrams)
Topics for the Final Exam

2. Turing machines

• basics of a TM, definition of TM computation, TM diagram,
• tracing a TM on a given input
• designing simple TMs (diagrams)
• equivalent models of TMs, non-deterministic TM
Topics for the Final Exam

2. Turing machines
   - basics of a TM, definition of TM computation, TM diagram,
   - tracing a TM on a given input
   - designing simple TMs (diagrams)
   - equivalent models of TMs, non-deterministic TM
Topics for the Final Exam

3. Decidability and undecidability proofs
   - decidable languages defined with learned computation models such as DFA, EQDFA, CFG, etc.
   - simulation techniques used for decidability proofs
   - undecidability proofs for A TM, HALT TM, etc.
   - countable sets, uncountable sets
   - diagonalisation technique, mapping reduction ≤m and properties

4. Computational complexity
   - conceptual only.

5. Proof by induction
Topics for the Final Exam

3. Decidability and undecidability proofs

- Decidable languages defined with learned computation models such as DFA, EQDFA, CFG, etc.
- Simulation techniques used for decidability proofs
- Undecidability proofs for A_TM, HALT_TM, etc.
- Countable sets, uncountable sets
- Diagonalisation technique, mapping reduction ≤_m and properties

4. Computational complexity

- Conceptual only.

5. Proof by induction
Topics for the Final Exam

3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
- simulation techniques used for decidability proofs
3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
- simulation techniques used for decidability proofs
- undecidability proofs for $A_{TM}$, $HALT_{TM}$, etc
Topics for the Final Exam

3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
- simulation techniques used for decidability proofs
- undecidability proofs for $A_{TM}$, $HALT_{TM}$, etc
  - countable sets, uncountable sets
3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
- simulation techniques used for decidability proofs
- undecidability proofs for $A_{TM}$, $HALT_{TM}$, etc
  - countable sets, uncountable sets
  - diagonalisation technique,
3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
- simulation techniques used for decidability proofs
- undecidability proofs for $A_{TM}$, $HALT_{TM}$, etc
  - countable sets, uncountable sets
  - diagonalisation technique,
  - mapping reduction $\leq_m$ and properties
3. Decidability and undecidability proofs

- decidable languages defined with learned computation models such as $A_{DFA}, EQ_{DFA}, E_{CFG},$ etc.
- simulation techniques used for decidability proofs
- undecidability proofs for $A_{TM}, HALT_{TM},$ etc
  - countable sets, uncountable sets
  - diagonalisation technique,
  - mapping reduction $\leq_m$ and properties

4. Computational complexity

- conceptual only.
Topics for the Final Exam

3. Decidability and undecidability proofs
   - decidable languages defined with learned computation models such as $A_{DFA}$, $EQ_{DFA}$, $E_{CFG}$, etc.
   - simulation techniques used for decidability proofs
   - undecidability proofs for $A_{TM}$, $HALT_{TM}$, etc
     - countable sets, uncountable sets
     - diagonalisation technique,
     - mapping reduction $\leq_m$ and properties

4. Computational complexity
   - conceptual only.

5. Proof by induction