Computability and Complexity
COSC 4200

Reducibility

Mapping Reducibility

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w halts with just $f(w)$ on its tape.

Mapping Reducibility

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w halts with just $f(w)$ on its tape.

Definition

Language A is *mapping reducible* to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B.$$

The function f is called a *reduction* of A to B.

Mapping Reducibility

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w halts with just $f(w)$ on its tape.

Definition

Language A is *mapping reducible* to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B.$$

The function f is called a *reduction* of A to B.

The reduction maps positive instances to positive instances:

$$w \in A \Rightarrow f(w) \in B$$

and negative instance to negative instances:

$$w \notin A \Rightarrow f(w) \notin B.$$

The reduction maps positive instances to positive instances:

$$w \in A \Rightarrow f(w) \in B$$

and negative instance to negative instances:

$$w \notin A \Rightarrow f(w) \notin B.$$

The reduction converts questions about membership in A to questions about membership in B.

The reduction maps positive instances to positive instances:

$$w \in A \Rightarrow f(w) \in B$$

and negative instance to negative instances:

$$w \notin A \Rightarrow f(w) \notin B.$$

The reduction converts questions about membership in A to questions about membership in B.

We can combine a reduction with an algorithm that decides B to obtain an algorithm that decides A.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Proof. Let f be a mapping reduction of A to B and let M be a decider for B. We describe a decider N for A.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Proof. Let f be a mapping reduction of A to B and let M be a decider for B. We describe a decider N for A.

N: On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$.
    - If M accepts $f(w)$, accept.
    - If M rejects $f(w)$, reject.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Proof. Let f be a mapping reduction of A to B and let M be a decider for B. We describe a decider N for A.

N: On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$.
   - If M accepts $f(w)$, accept.
   - If M rejects $f(w)$, reject.

We claim that N decides A.

- If $w \in A$, then $f(w) \in B$ because f reduces A to B. Therefore M accepts $f(w)$ and N accepts w.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Proof. Let f be a mapping reduction of A to B and let M be a decider for B. We describe a decider N for A.

N: On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$.
    - If M accepts $f(w)$, accept.
    - If M rejects $f(w)$, reject.

We claim that N decides A.

- If $w \in A$, then $f(w) \in B$ because f reduces A to B. Therefore M accepts $f(w)$ and N accepts w.
- If $w \notin A$, then $f(w) \notin B$ because f reduces A to B. Therefore M rejects $f(w)$ and N rejects w. $\square$

Theorem

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

This provides a tool for proving undecidability:

Corollary

If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

This provides a tool for proving undecidability:

Corollary

If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Proof. Assume the hypothesis and suppose that B is decidable. Then A is decidable by the theorem above, a contradiction. $\square$

Halting Problem for TMs

The *halting problem* for TMs:

$$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that halts input string } w \}.$$

Theorem

$HALT_{\mathrm{TM}}$ is undecidable.

Second proof. We proved this earlier, but now we give a proof by showing that $A_{\mathrm{TM}} \leq_{\mathrm{m}} HALT_{\mathrm{TM}}$.

Halting Problem for TMs

The *halting problem* for TMs:

$$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that halts input string } w\ \}.$$

Theorem

$HALT_{\mathrm{TM}}$ is undecidable.

Second proof. We proved this earlier, but now we give a proof by showing that $A_{\mathrm{TM}} \leq_{\mathrm{m}} HALT_{\mathrm{TM}}$.

We must design a computable function f so that for any instance $\langle M, w\rangle$ of A_{TM}, $f(\langle M, w\rangle) = \langle M', w'\rangle$ so that

$$\langle M, w\rangle \in A_{\mathrm{TM}} \Leftrightarrow \langle M', w'\rangle \in HALT_{\mathrm{TM}}.$$

The following Turing machine F computes a reduction f.

F: On input $\langle M, w\rangle$:

1. Construct the following machine M':
   - M': On input x:
     1. Run M on x.
     2. If M accepts, accept.
     3. If M rejects, enter an infinite loop.
2. Output $\langle M', w\rangle$.

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\langle M, w\rangle \in A_{\mathrm{TM}} \quad \Rightarrow \quad M \text{ accepts } w$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned} \langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\ &\Rightarrow M' \text{ accepts } w \end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w
\end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

M rejects w

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} \quad &\Rightarrow \quad M \text{ accepts } w \\
&\Rightarrow \quad M' \text{ accepts } w \\
&\Rightarrow \quad M' \text{ halts on } w \\
&\Rightarrow \quad \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$M \text{ rejects } w \quad \Rightarrow \quad M' \text{ on } w \text{ goes into an infinite loop in step } ③$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w
\end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

$$M \text{ does not halt on } w$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

$$M \text{ does not halt on } w \Rightarrow M' \text{ simulates } M \text{ on } w \text{ forever in step } ①$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

$$\begin{aligned}
M \text{ does not halt on } w &\Rightarrow M' \text{ simulates } M \text{ on } w \text{ forever in step } ① \\
&\Rightarrow M' \text{ does not halt on } w
\end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

$$\begin{aligned}
M \text{ does not halt on } w &\Rightarrow M' \text{ simulates } M \text{ on } w \text{ forever in step } ① \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

We claim that f is a mapping reduction of A_{TM} to $HALT_{\mathrm{TM}}$.

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow M \text{ accepts } w \\
&\Rightarrow M' \text{ accepts } w \\
&\Rightarrow M' \text{ halts on } w \\
&\Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}
\end{aligned}$$

If $\langle M, w\rangle \notin A_{\mathrm{TM}}$, then either M rejects w or M does not halt on w. We consider these two cases separately.

$$\begin{aligned}
M \text{ rejects } w &\Rightarrow M' \text{ on } w \text{ goes into an infinite loop in step } ③ \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

$$\begin{aligned}
M \text{ does not halt on } w &\Rightarrow M' \text{ simulates } M \text{ on } w \text{ forever in step } ① \\
&\Rightarrow M' \text{ does not halt on } w \\
&\Rightarrow \langle M', w\rangle \notin HALT_{\mathrm{TM}}
\end{aligned}$$

Therefore $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow \langle M', w\rangle \in HALT_{\mathrm{TM}}$.

We have proved

$$\langle M, w\rangle \in A_{\mathrm{TM}} \Leftrightarrow f(M, w) = \langle M', w\rangle \in \mathit{HALT}_{\mathrm{TM}},$$

so $A_{\mathrm{TM}} \leq_{\mathrm{m}} \mathit{HALT}_{\mathrm{TM}}$ via f. Since A_{TM} is undecidable, it follows that $\mathit{HALT}_{\mathrm{TM}}$ is undecidable. $\square$

Regularity Problem for TMs

$$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}.$$

Theorem

$REGULAR_{\mathrm{TM}}$ is undecidable.

Regularity Problem for TMs

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}.$

Theorem

$REGULAR_{\mathrm{TM}}$ is undecidable.

Proof. We will show that $A_{\mathrm{TM}} \leq_{\mathrm{m}} REGULAR_{\mathrm{TM}}$. Given a TM M and a string w, define a TM $M_{(w)}$ as follows:

$M_{(w)}$: On input x:

1. If x has the form $0^n 1^n$, accept.
2. If x does not have this form, run M on input w and accept if M accepts w.

Regularity Problem for TMs

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}.$

Theorem

$REGULAR_{\mathrm{TM}}$ is undecidable.

Proof. We will show that $A_{\mathrm{TM}} \leq_{\mathrm{m}} REGULAR_{\mathrm{TM}}$. Given a TM M and a string w, define a TM $M_{(w)}$ as follows:

$M_{(w)}$: On input x:

1. If x has the form $0^n 1^n$, accept.
2. If x does not have this form, run M on input w and accept if M accepts w.

- If M accepts w, then $L(M_{(w)}) = \Sigma^*$, so $L(M_{(w)})$ is regular.

Regularity Problem for TMs

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$.

Theorem

$REGULAR_{\mathrm{TM}}$ is undecidable.

Proof. We will show that $A_{\mathrm{TM}} \leq_{\mathrm{m}} REGULAR_{\mathrm{TM}}$. Given a TM M and a string w, define a TM $M_{(w)}$ as follows:

$M_{(w)}$: On input x:

1. If x has the form $0^n 1^n$, accept.
2. If x does not have this form, run M on input w and accept if M accepts w.

- If M accepts w, then $L(M_{(w)}) = \Sigma^*$, so $L(M_{(w)})$ is regular.
- If M does not accept w, then $L(M_{(w)}) = \{0^n 1^n \mid n \geq 0\}$, so $L(M_{(w)})$ is not regular.

We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}\rangle.$$

Then f is computable,

We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}\rangle.$$

Then f is computable,

$$\begin{aligned} \langle M, w\rangle \in A_{\mathrm{TM}} \quad &\Rightarrow \quad L(M_{(w)}) = \Sigma^* \\ &\Rightarrow \quad \langle M_{(w)}\rangle \in \mathit{REGULAR}_{\mathrm{TM}}, \end{aligned}$$

We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}\rangle.$$

Then f is computable,

$$\begin{aligned}\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow \langle M_{(w)}\rangle \in \mathit{REGULAR}_{\mathrm{TM}},\end{aligned}$$

and

$$\begin{aligned}\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \{0^n 1^n \mid n \geq 0\} \\ &\Rightarrow \langle M_{(w)}\rangle \notin \mathit{REGULAR}_{\mathrm{TM}},\end{aligned}$$

We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}\rangle.$$

Then f is computable,

$$\begin{aligned}\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow \langle M_{(w)}\rangle \in \mathit{REGULAR}_{\mathrm{TM}},\end{aligned}$$

and

$$\begin{aligned}\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \{0^n 1^n \mid n \geq 0\} \\ &\Rightarrow \langle M_{(w)}\rangle \notin \mathit{REGULAR}_{\mathrm{TM}},\end{aligned}$$

so $A_{\mathrm{TM}} \leq_{\mathrm{m}} \mathit{REGULAR}_{\mathrm{TM}}$ via f.

Since A_{TM} is undecidable, it follows that $\mathit{REGULAR}_{\mathrm{TM}}$ is undecidable. $\square$

Equivalence Problem for TMs

$$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\ \}$$

Theorem

EQ_{TM} is undecidable.

Equivalence Problem for TMs

$$EQ_{\mathrm{TM}} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\ \}$$

Theorem

EQ_{TM} is undecidable.

Proof. We will reduce

$$E_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

to EQ_{TM}.

Equivalence Problem for TMs

$$EQ_{\mathrm{TM}} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\ \}$$

Theorem

EQ_{TM} is undecidable.

Proof. We will reduce

$$E_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

to EQ_{TM}.

Let R be a TM that immediately rejects all inputs. Then $L(R) = \emptyset$. Our reduction f is defined by

$$f(\langle M \rangle) = \langle M, R \rangle.$$

Then

$$\langle M \rangle \in E_{\mathrm{TM}} \quad \Leftrightarrow \quad L(M) = \emptyset$$

Then

$$\begin{aligned} \langle M \rangle \in E_{\mathrm{TM}} \quad &\Leftrightarrow \quad L(M) = \emptyset \\ &\Leftrightarrow \quad L(M) = L(R) \end{aligned}$$

Then

$$\begin{aligned} \langle M \rangle \in E_{\mathrm{TM}} &\Leftrightarrow L(M) = \emptyset \\ &\Leftrightarrow L(M) = L(R) \\ &\Leftrightarrow \langle M, R \rangle \in EQ_{\mathrm{TM}} \end{aligned}$$

Then

$$\begin{aligned}
\langle M \rangle \in E_{\mathrm{TM}} &\Leftrightarrow L(M) = \emptyset \\
&\Leftrightarrow L(M) = L(R) \\
&\Leftrightarrow \langle M, R \rangle \in EQ_{\mathrm{TM}} \\
&\Leftrightarrow f(\langle M \rangle) \in EQ_{\mathrm{TM}},
\end{aligned}$$

Then

$$\begin{aligned}
\langle M\rangle \in E_{\mathrm{TM}} &\Leftrightarrow L(M) = \emptyset \\
&\Leftrightarrow L(M) = L(R) \\
&\Leftrightarrow \langle M, R\rangle \in EQ_{\mathrm{TM}} \\
&\Leftrightarrow f(\langle M\rangle) \in EQ_{\mathrm{TM}},
\end{aligned}$$

so $E_{\mathrm{TM}} \leq_{\mathrm{m}} EQ_{\mathrm{TM}}$ via f. Since E_{TM} is undecidable, it follows that EQ_{TM} is undecidable. $\square$

In fact, we can even show that EQ_{TM} is not Turing-recognizable or co-Turing-recognizable. For this, we need the following theorem.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof. Let f be a mapping reduction of A to B and let M be a recognizer for B. We describe a recognizer N for A.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof. Let f be a mapping reduction of A to B and let M be a recognizer for B. We describe a recognizer N for A.

N: On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$.
    - If M accepts $f(w)$, accept.
    - If M rejects $f(w)$, reject.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof. Let f be a mapping reduction of A to B and let M be a recognizer for B. We describe a recognizer N for A.

N: On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$.
   - If M accepts $f(w)$, accept.
   - If M rejects $f(w)$, reject.

We claim that N recognizes A.

- If $w \in A$, then $f(w) \in B$ because f reduces A to B. Therefore M accepts $f(w)$ and N accepts w.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof. Let f be a mapping reduction of A to B and let M be a recognizer for B. We describe a recognizer N for A.

N: On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$.
   - If M accepts $f(w)$, accept.
   - If M rejects $f(w)$, reject.

We claim that N recognizes A.

- If $w \in A$, then $f(w) \in B$ because f reduces A to B. Therefore M accepts $f(w)$ and N accepts w.
- If $w \notin A$, then $f(w) \notin B$ because f reduces A to B. Therefore M either rejects $f(w)$ or does not halt on $f(w)$ and N does the same on w. $\square$

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Corollary

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Theorem

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Corollary

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Corollary

If $A \leq_{\mathrm{m}} B$ and A is not co-Turing-recognizable, then B is not co-Turing-recognizable.

Proof. Use the fact $A \leq_{\mathrm{m}} B \iff A^c \leq_{\mathrm{m}} B^c$ and the previous corollary. $\square$

Theorem

EQ_{TM} is not Turing-recognizable.

Proof. We showed that:

- E_{TM} is not Turing-recognizable.
- $E_{\mathrm{TM}} \leq_{\mathrm{m}} EQ_{\mathrm{TM}}$.

The previous theorem implies that EQ_{TM} is not Turing-recognizable. $\square$

Theorem

EQ_{TM} is not co-Turing-recognizable.

Theorem

EQ_{TM} is not co-Turing-recognizable.

Proof. We will reduce A_{TM} to EQ_{TM}. Since A_{TM} is not co-Turing-recognizable, this will establish that EQ_{TM} is not co-Turing-recognizable as well.

For any TM M and string w, define a TM $M_{(w)}$ as follows.

$M_{(w)}$: On input x:

1. Run M on input w. // Ignore input x.
2. If M accepts w, accept x.

For any TM M and string w, define a TM $M_{(w)}$ as follows.

$M_{(w)}$: On input x:

1. Run M on input w. // Ignore input x.
2. If M accepts w, accept x.

Consider any instance $\langle M, w \rangle$ of A_{TM}.

- Suppose $\langle M, w \rangle \in A_{\mathrm{TM}}$. Then M accepts w. Therefore $L(M_{(w)}) = \Sigma^*$.

For any TM M and string w, define a TM $M_{(w)}$ as follows.

$M_{(w)}$: On input x:

1. Run M on input w. // Ignore input x.
2. If M accepts w, accept x.

Consider any instance $\langle M, w \rangle$ of A_{TM}.

- Suppose $\langle M, w \rangle \in A_{\mathrm{TM}}$. Then M accepts w. Therefore $L(M_{(w)}) = \Sigma^*$.
- Suppose $\langle M, w \rangle \notin A_{\mathrm{TM}}$. Then M does not accept w. Therefore $L(M_{(w)}) = \emptyset$.

For any TM M and string w, define a TM $M_{(w)}$ as follows.

$M_{(w)}$: On input x:

1. Run M on input w. // Ignore input x.
2. If M accepts w, accept x.

Consider any instance $\langle M, w \rangle$ of A_{TM}.

- Suppose $\langle M, w \rangle \in A_{\mathrm{TM}}$. Then M accepts w. Therefore $L(M_{(w)}) = \Sigma^*$.
- Suppose $\langle M, w \rangle \notin A_{\mathrm{TM}}$. Then M does not accept w. Therefore $L(M_{(w)}) = \emptyset$.

Therefore we have

- $\langle M, w \rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w \rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

We have

- $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}, T\rangle.$$

We have

- $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}, T\rangle.$$

Then f is computable,

We have

- $\langle M, w \rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w \rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w \rangle) = \langle M_{(w)}, T \rangle.$$

Then f is computable,

$$\langle M, w \rangle \in A_{\mathrm{TM}} \quad \Rightarrow \quad L(M_{(w)}) = \Sigma^*$$

We have

- $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}, T\rangle.$$

Then f is computable,

$$\begin{aligned} \langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow L(M_{(w)}) = L(T) \end{aligned}$$

We have

- $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}, T\rangle.$$

Then f is computable,

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} \quad &\Rightarrow \quad L(M_{(w)}) = \Sigma^* \\
&\Rightarrow \quad L(M_{(w)}) = L(T) \\
&\Rightarrow \quad \langle M_{(w)}, T\rangle \in EQ_{\mathrm{TM}}
\end{aligned}$$

We have

- $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}, T\rangle.$$

Then f is computable,

$$\begin{aligned} \langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow L(M_{(w)}) = L(T) \\ &\Rightarrow \langle M_{(w)}, T\rangle \in EQ_{\mathrm{TM}} \end{aligned}$$

and

$$\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$$

We have

- $\langle M, w \rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w \rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w \rangle) = \langle M_{(w)}, T \rangle.$$

Then f is computable,

$$\begin{aligned} \langle M, w \rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow L(M_{(w)}) = L(T) \\ &\Rightarrow \langle M_{(w)}, T \rangle \in EQ_{\mathrm{TM}} \end{aligned}$$

and

$$\begin{aligned} \langle M, w \rangle \notin A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \emptyset \\ &\Rightarrow L(M_{(w)}) \neq L(T) \end{aligned}$$

We have

- $\langle M, w \rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w \rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w \rangle) = \langle M_{(w)}, T \rangle.$$

Then f is computable,

$$\begin{aligned} \langle M, w \rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow L(M_{(w)}) = L(T) \\ &\Rightarrow \langle M_{(w)}, T \rangle \in EQ_{\mathrm{TM}} \end{aligned}$$

and

$$\begin{aligned} \langle M, w \rangle \notin A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \emptyset \\ &\Rightarrow L(M_{(w)}) \neq L(T) \\ &\Rightarrow \langle M_{(w)}, T \rangle \notin EQ_{\mathrm{TM}}, \end{aligned}$$

We have

- $\langle M, w\rangle \in A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \Sigma^*$
- $\langle M, w\rangle \notin A_{\mathrm{TM}} \Rightarrow L(M_{(w)}) = \emptyset$.

Let T be a TM with $L(T) = \Sigma^*$. We define our reduction f as

$$f(\langle M, w\rangle) = \langle M_{(w)}, T\rangle.$$

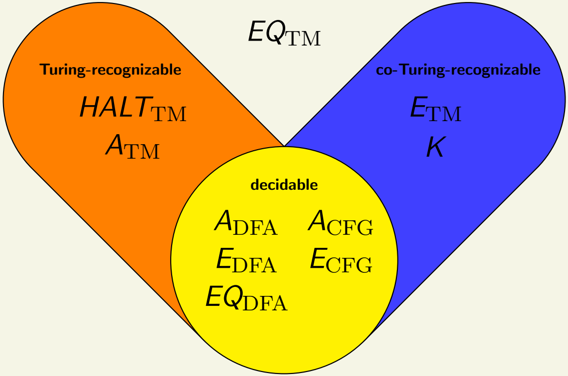
Then f is computable,

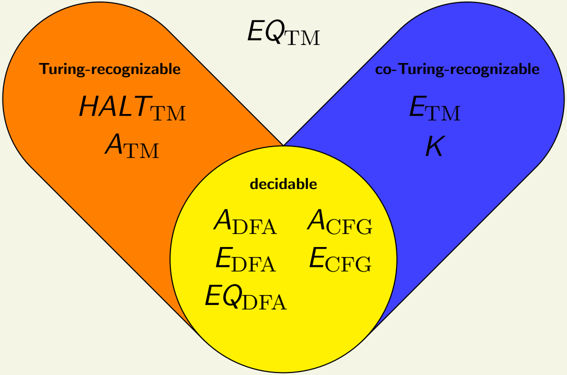
$$\begin{aligned} \langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \Sigma^* \\ &\Rightarrow L(M_{(w)}) = L(T) \\ &\Rightarrow \langle M_{(w)}, T\rangle \in EQ_{\mathrm{TM}} \end{aligned}$$

and

$$\begin{aligned} \langle M, w\rangle \notin A_{\mathrm{TM}} &\Rightarrow L(M_{(w)}) = \emptyset \\ &\Rightarrow L(M_{(w)}) \neq L(T) \\ &\Rightarrow \langle M_{(w)}, T\rangle \notin EQ_{\mathrm{TM}}, \end{aligned}$$

so $A_{\mathrm{TM}} \leq_{\mathrm{m}} EQ_{\mathrm{TM}}$ via f. $\square$


EQ_{TM}
Turing-recognizable
$HALT_{\mathrm{TM}}$
A_{TM}
co-Turing-recognizable
E_{TM}
K
decidable
A_{DFA}
A_{CFG}
E_{DFA}
E_{CFG}
EQ_{DFA}



What about EQ_{CFG}?

Define

$$ALL_{\mathrm{DFA}} = \{\langle M \rangle \mid M \text{ is a DFA and } L(M) = \Sigma^*\},$$

$$ALL_{\mathrm{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}.$$

Define

$$ALL_{\mathrm{DFA}} = \{\langle M\rangle \mid M \text{ is a DFA and } L(M) = \Sigma^*\},$$

$$ALL_{\mathrm{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}.$$

Theorem

$\mathrm{ALL}_{\mathrm{DFA}}$ *is decidable.*

Proof. Use similar techniques to E_{DFA} or reduce to EQ_{DFA}. $\square$

Define

$$ALL_{\mathrm{DFA}} = \{\langle M\rangle \mid M \text{ is a DFA and } L(M) = \Sigma^*\},$$

$$ALL_{\mathrm{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}.$$

Theorem

$\mathrm{ALL}_{\mathrm{DFA}}$ *is decidable.*

Proof. Use similar techniques to E_{DFA} or reduce to EQ_{DFA}. □

We will show that ALL_{CFG} is undecidable and as a corollary, that EQ_{CFG} is also undecidable.

$$EQ_{\mathrm{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\ \}$$

Definition

Let M be a TM and w be a string.

- An *accepting computation history* of M on w is a sequence of configurations $C_1, \ldots, C_l$, where C_1 is the start configuration of M on w, C_l is an accepting configuration of M, and each C_i follows from C_{i-1} using M's transition function.
- A *rejection computation history* is defined similarly, except that C_l is a rejecting configuration.

Theorem

ALL_{CFG} is undecidable.

Proof. We will show that $A^{c}_{\mathrm{TM}} \leq_{\mathrm{m}} ALL_{\mathrm{CFG}}$. The proof uses computation histories.

Theorem

ALL_{CFG} is undecidable.

Proof. We will show that $A_{\mathrm{TM}}^c \leq_{\mathrm{m}} ALL_{\mathrm{CFG}}$. The proof uses computation histories.

Given a TM and a string w, our goal is to construct a CFG G so that

- M does not accept $w \Rightarrow L(G) = \Sigma^*$,
- M accepts $w \Rightarrow L(G) \neq \Sigma^*$.

Theorem

ALL_{CFG} is undecidable.

Proof. We will show that $A^c_{\mathrm{TM}} \leq_{\mathrm{m}} ALL_{\mathrm{CFG}}$. The proof uses computation histories.

Given a TM and a string w, our goal is to construct a CFG G so that

- M does not accept $w \Rightarrow L(G) = \Sigma^*$,
- M accepts $w \Rightarrow L(G) \neq \Sigma^*$.

The idea is to design G sot that G generates all strings that do not encode accepting computation histories of M on w.

The natural way to encode a computation history $C_1, \ldots, C_l$ is as a string

$$\#C_1\#C_2\#C_3\#\cdots\#C_l\#.$$

However, we will encode the computation history as

$$\#C_1\#C_2^R\#C_3\#C_4^R\#\cdots\#C_l\#,$$

with every other configuration written in reverse.

The natural way to encode a computation history $C_1, \dots, C_l$ is as a string

$$\#C_1\#C_2\#C_3\#\cdots\#C_l\#.$$

However, we will encode the computation history as

$$\#C_1\#C_2^R\#C_3\#C_4^R\#\cdots\#C_l\#,$$

with every other configuration written in reverse.

The language of our grammar will be

$$L(G) = \left\{ x \in \Sigma^* \;\middle|\; \begin{array}{l} x \text{ is not an accepting computation} \\ \text{history of } M \text{ on } w \end{array} \right\}.$$

$$L(G) = \left\{ x \in \Sigma^* \;\middle|\; \begin{array}{l} x \text{ is not an accepting computation} \\ \text{history of } M \text{ on } w \end{array} \right\}.$$

That is, G generates all strings

$$x = \#C_1\#C_2^R\#C_3\#C_4^R \cdots \#C_l\#$$

such that:

1. C_1 is not the start configuration of M on w,

$$L(G) = \left\{ x \in \Sigma^* \;\middle|\; \begin{array}{l} x \text{ is not an accepting computation} \\ \text{history of } M \text{ on } w \end{array} \right\}.$$

That is, G generates all strings

$$x = \#C_1\#C_2^R\#C_3\#C_4^R \cdots \#C_l\#$$

such that:

1. C_1 is not the start configuration of M on w,
2. C_l is not an accepting configuration, or

$$L(G) = \left\{ x \in \Sigma^* \;\middle|\; \begin{array}{l} x \text{ is not an accepting computation} \\ \text{history of } M \text{ on } w \end{array} \right\}.$$

That is, G generates all strings

$$x = \#C_1\#C_2^R\#C_3\#C_4^R \cdots \#C_l\#$$

such that:

1. C_1 is not the start configuration of M on w,
2. C_l is not an accepting configuration, or
3. for some i, C_i does not yield C_{i+1} under M's transition function

$$L(G) = \left\{ x \in \Sigma^* \;\middle|\; \begin{array}{l} x \text{ is not an accepting computation} \\ \text{history of } M \text{ on } w \end{array} \right\}.$$

That is, G generates all strings

$$x = \#C_1\#C_2^R\#C_3\#C_4^R \cdots \#C_l\#$$

such that:

1. C_1 is not the start configuration of M on w,
2. C_l is not an accepting configuration, or
3. for some i, C_i does not yield C_{i+1} under M's transition function

as well as all x where

4. x is not of the form $\#C_1\#C_2^R\#C_3\#C_4^R \cdots \#C_l\#$.

To make things simpler, we design a PDA D first. An algorithm can convert D into G.

To make things simpler, we design a PDA D first. An algorithm can convert D into G.

D starts by nondeterministically choosing one of the 4 conditions to check. Conditions ❶, ❷, and ❹ are easy to check (can be done on a DFA).

To make things simpler, we design a PDA D first. An algorithm can convert D into G.

D starts by nondeterministically choosing one of the 4 conditions to check. Conditions ❶, ❷, and ❹ are easy to check (can be done on a DFA).

For condition ❸:

- D nondeterministically chooses two configurations C_i and C_{i+1} to check.

To make things simpler, we design a PDA D first. An algorithm can convert D into G.

D starts by nondeterministically choosing one of the 4 conditions to check. Conditions ❶, ❷, and ❹ are easy to check (can be done on a DFA).

For condition ❸:

- D nondeterministically chooses two configurations C_i and C_{i+1} to check.
- Then D pushes C_i onto its stack, and compares it to C_{i+1}. This is why we need the encoding of computation histories that writes every other configuration in reverse – C_i will be pushed onto the stack in reverse.

To make things simpler, we design a PDA D first. An algorithm can convert D into G.

D starts by nondeterministically choosing one of the 4 conditions to check. Conditions ❶, ❷, and ❹ are easy to check (can be done on a DFA).

For condition ❸:

- D nondeterministically chooses two configurations C_i and C_{i+1} to check.
- Then D pushes C_i onto its stack, and compares it to C_{i+1}. This is why we need the encoding of computation histories that writes every other configuration in reverse – C_i will be pushed onto the stack in reverse.
- Now D can compare the two configurations, they should match except for the three cells around the head position of C_i.

To make things simpler, we design a PDA D first. An algorithm can convert D into G.

D starts by nondeterministically choosing one of the 4 conditions to check. Conditions ❶, ❷, and ❹ are easy to check (can be done on a DFA).

For condition ❸:

- D nondeterministically chooses two configurations C_i and C_{i+1} to check.
- Then D pushes C_i onto its stack, and compares it to C_{i+1}. This is why we need the encoding of computation histories that writes every other configuration in reverse – C_i will be pushed onto the stack in reverse.
- Now D can compare the two configurations, they should match except for the three cells around the head position of C_i.
- D accepts if there is a mismatch. $\square$

The proof that ALL_{CFG} is undecidable reduces $A^c_{\mathrm{TM}} \leq_{\mathrm{m}} ALL_{\mathrm{CFG}}$. Because A^c_{TM} is not Turing-recognizable, we actually have a stronger result:

Theorem

ALL_{CFG} is not Turing-recognizable.

The proof that ALL_{CFG} is undecidable reduces $A^c_{\mathrm{TM}} \leq_{\mathrm{m}} ALL_{\mathrm{CFG}}$. Because A^c_{TM} is not Turing-recognizable, we actually have a stronger result:

Theorem

ALL_{CFG} is not Turing-recognizable.

However:

Theorem

ALL_{CFG} is co-Turing-recognizable.

Theorem

ALL_{CFG} is co-Turing-recognizable.

Proof. We need to give a recognition algorithm for ALL^c_{CFG}. Let $s_1, s_2, \ldots$ be an enumeration of Σ^*.

> M: On input $\langle G \rangle$:
>
> for $i = 1, 2, \ldots$
>
> Run the decision algorithm S for A_{CFG} on input $\langle G, s_i \rangle$.
>
> If it rejects, accept.

Theorem

ALL_{CFG} is co-Turing-recognizable.

Proof. We need to give a recognition algorithm for ALL^c_{CFG}. Let $s_1, s_2, \ldots$ be an enumeration of Σ^*.

> M: On input $\langle G \rangle$:
>
> for $i = 1, 2, \ldots$
>
> Run the decision algorithm S for A_{CFG} on input $\langle G, s_i \rangle$.
>
> If it rejects, accept.

Suppose that $\langle G \rangle \notin ALL_{\mathrm{CFG}}$. Then $L(G) \neq \Sigma^*$. Let s_j be the least string that is not in $L(G)$. Then on the j^{th} iteration of the loop, S will reject $\langle G, s_j \rangle$. Therefore M will accept $\langle G \rangle$.

Theorem

ALL_{CFG} is co-Turing-recognizable.

Proof. We need to give a recognition algorithm for ALL^c_{CFG}. Let $s_1, s_2, \ldots$ be an enumeration of Σ^*.

> M: On input $\langle G \rangle$:
>
> for $i = 1, 2, \ldots$
> Run the decision algorithm S for A_{CFG} on input $\langle G, s_i \rangle$.
> If it rejects, accept.

Suppose that $\langle G \rangle \notin ALL_{\mathrm{CFG}}$. Then $L(G) \neq \Sigma^*$. Let s_j be the least string that is not in $L(G)$. Then on the j^{th} iteration of the loop, S will reject $\langle G, s_j \rangle$. Therefore M will accept $\langle G \rangle$.

Now suppose that $\langle G \rangle \in ALL_{\mathrm{CFG}}$. Then $L(G) = \Sigma^*$, so $\langle G, s_i \rangle \in A_{\mathrm{CFG}}$ for all i. Thus S will accept in every iteration and M will run forever on $\langle G \rangle$.

Theorem

ALL_{CFG} is co-Turing-recognizable.

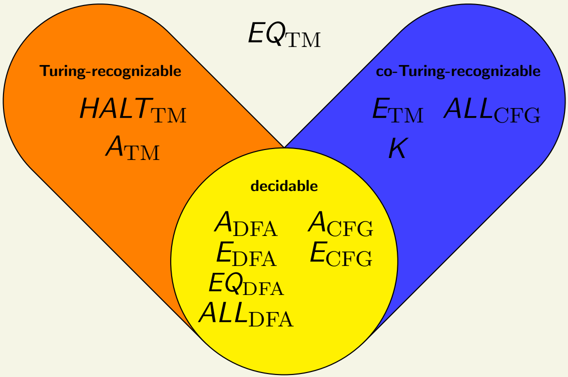
Proof. We need to give a recognition algorithm for ALL^c_{CFG}. Let $s_1, s_2, \ldots$ be an enumeration of Σ^*.

> M: On input $\langle G \rangle$:
>
> for $i = 1, 2, \ldots$
>
> Run the decision algorithm S for A_{CFG} on input $\langle G, s_i \rangle$.
>
> If it rejects, accept.

Suppose that $\langle G \rangle \notin ALL_{\mathrm{CFG}}$. Then $L(G) \neq \Sigma^*$. Let s_j be the least string that is not in $L(G)$. Then on the j^{th} iteration of the loop, S will reject $\langle G, s_j \rangle$. Therefore M will accept $\langle G \rangle$.

Now suppose that $\langle G \rangle \in ALL_{\mathrm{CFG}}$. Then $L(G) = \Sigma^*$, so $\langle G, s_i \rangle \in A_{\mathrm{CFG}}$ for all i. Thus S will accept in every iteration and M will run forever on $\langle G \rangle$.

Therefore M recognizes ALL^c_{CFG}. $\square$


EQ_{TM}
Turing-recognizable
$HALT_{\mathrm{TM}}$
A_{TM}
co-Turing-recognizable
E_{TM}
ALL_{CFG}
K
decidable
A_{DFA}
A_{CFG}
E_{DFA}
E_{CFG}
EQ_{DFA}
ALL_{DFA}

Recall the *equivalence problem* for CFGs:

$$EQ_{\mathrm{CFG}} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \}$$

Theorem

EQ_{CFG} is undecidable.

Recall the *equivalence problem* for CFGs:

$$EQ_{\mathrm{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\ \}$$

Theorem

EQ_{CFG} is undecidable.

Proof. Show that $ALL_{\mathrm{CFG}} \leq_{\mathrm{m}} EQ_{\mathrm{CFG}}$. $\square$

Recall the *equivalence problem* for CFGs:

$$EQ_{\mathrm{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\ \}$$

Theorem

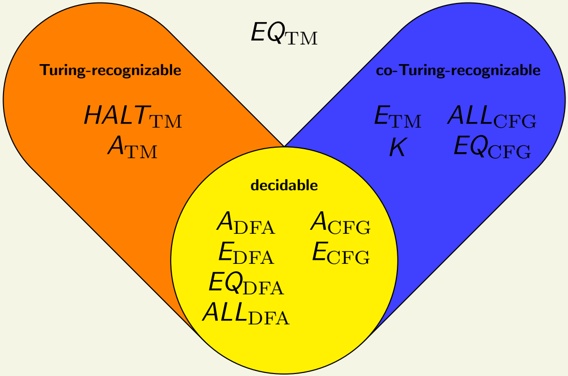
EQ_{CFG} is undecidable.

Proof. Show that $ALL_{\mathrm{CFG}} \leq_{\mathrm{m}} EQ_{\mathrm{CFG}}$. □

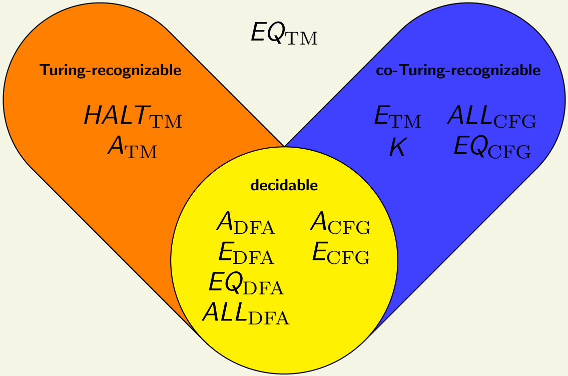
Theorem

EQ_{CFG} is co-Turing-recognizable.

Proof. Similar to ALL_{CFG} is co-Turing-recognizable. □



We have now classified the acceptance problem, emptiness problem, and equivalence problem for each of DFAs, CFGs, and TMs.



We have now classified the acceptance problem, emptiness problem, and equivalence problem for each of DFAs, CFGs, and TMs.

What about ALL_{TM}?

$$ALL_{\mathrm{TM}} = \{\langle M \rangle \mid G \text{ is a TM and } L(M) = \Sigma^*\}$$

Theorem

ALL_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

$$ALL_{\mathrm{TM}} = \{\langle M\rangle \mid G \text{ is a TM and } L(M) = \Sigma^*\}$$

Theorem

ALL_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

Proof. It suffices to show that $EQ_{\mathrm{TM}} \leq_{\mathrm{m}} ALL_{\mathrm{TM}}$, since EQ_{TM} is not Turing-recognizable and not co-Turing-recognizable.

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

M: On input x:

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

M: On input x:
 if $x \in 0^*$
  accept x

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

M: On input x:
    if $x \in 0^*$
        accept x
    else
        $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

M: On input x:
if $x \in 0^*$
  accept x
else
  $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$
  run M_1 on w for up to t steps
  run M_2 on w for up to t steps

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

M: On input x:
if $x \in 0^*$
  accept x
else
  $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$
  run M_1 on w for up to t steps
  run M_2 on w for up to t steps
  if M_1 accepts w within t steps
    run M_2 on w // for unlimited steps
    if M_2 accepts w, accept x
    if M_2 rejects w, reject x

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

```
M:  On input x:
    if x ∈ 0*
        accept x
    else
        x = 0^t 1w for some w ∈ Σ* and t ≥ 0
        run M1 on w for up to t steps
        run M2 on w for up to t steps
        if M1 accepts w within t steps
            run M2 on w // for unlimited steps
            if M2 accepts w, accept x
            if M2 rejects w, reject x
        else if M2 accepts w within t steps
            run M1 on w // for unlimited steps
            if M1 accepts w, accept x
            if M1 rejects w, reject x
```

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

```
M:  On input x:
    if x ∈ 0*
        accept x
    else
        x = 0^t 1w for some w ∈ Σ* and t ≥ 0
        run M1 on w for up to t steps
        run M2 on w for up to t steps
        if M1 accepts w within t steps
            run M2 on w // for unlimited steps
            if M2 accepts w, accept x
            if M2 rejects w, reject x
        else if M2 accepts w within t steps
            run M1 on w // for unlimited steps
            if M1 accepts w, accept x
            if M1 rejects w, reject x
        else
            accept x
```

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

```
M:  On input x:
    if x ∈ 0*
        accept x
    else
        x = 0^t 1w for some w ∈ Σ* and t ≥ 0
        run M1 on w for up to t steps
        run M2 on w for up to t steps
        if M1 accepts w within t steps
            run M2 on w // for unlimited steps
            if M2 accepts w, accept x
            if M2 rejects w, reject x
        else if M2 accepts w within t steps
            run M1 on w // for unlimited steps
            if M1 accepts w, accept x
            if M1 rejects w, reject x
        else
            accept x
```

Let $\langle M_1, M_2 \rangle$ be an instance of EQ_{TM}. Construct the following TM M:

```
M:  On input x:
    if x ∈ 0*
        accept x
    else
        x = 0^t 1w for some w ∈ Σ* and t ≥ 0
        run M1 on w for up to t steps
        run M2 on w for up to t steps
        if M1 accepts w within t steps
            run M2 on w // for unlimited steps
            if M2 accepts w, accept x
            if M2 rejects w, reject x
        else if M2 accepts w within t steps
            run M1 on w // for unlimited steps
            if M1 accepts w, accept x
            if M1 rejects w, reject x
        else
            accept x
```

We claim that:

$$L(M_1) = L(M_2) \quad \Rightarrow \quad L(M) = \Sigma^*,$$
$$L(M_1) \neq L(M_2) \quad \Rightarrow \quad L(M) \neq \Sigma^*.$$

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

- If $x \in 0^*$, then M accepts x.

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

- If $x \in 0^*$, then M accepts x.
- Otherwise $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$.

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

- If $x \in 0^*$, then M accepts x.
- Otherwise $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$.
  - If M_1 accepts w within t steps, then M will run M_2 on w.
    Since $L(M_1) = L(M_2)$, M_2 will accept w.
    Therefore M accepts x.

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

- If $x \in 0^*$, then M accepts x.
- Otherwise $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$.
  - If M_1 accepts w within t steps, then M will run M_2 on w.
    Since $L(M_1) = L(M_2)$, M_2 will accept w.
    Therefore M accepts x.
  - Else if M_2 accepts w within t steps, then M will run M_1 on w.
    Since $L(M_1) = L(M_2)$, M_1 will accept w.
    Therefore M accepts x.

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

- If $x \in 0^*$, then M accepts x.
- Otherwise $x = 0^t 1w$ for some $w \in \Sigma^*$ and $t \geq 0$.
  - If M_1 accepts w within t steps, then M will run M_2 on w.
    Since $L(M_1) = L(M_2)$, M_2 will accept w.
    Therefore M accepts x.
  - Else if M_2 accepts w within t steps, then M will run M_1 on w.
    Since $L(M_1) = L(M_2)$, M_1 will accept w.
    Therefore M accepts x.
  - Otherwise, M_1 does not accept w within t steps and M_2 does not accept w within t steps. Then M accepts x.

Suppose $L(M_1) = L(M_2)$. We want to show $L(M) = \Sigma^*$.
Let $x \in \Sigma^*$.

- If $x \in 0^*$, then M accepts x.
- Otherwise $x = 0^t 1 w$ for some $w \in \Sigma^*$ and $t \geq 0$.
  - If M_1 accepts w within t steps, then M will run M_2 on w. Since $L(M_1) = L(M_2)$, M_2 will accept w. Therefore M accepts x.
  - Else if M_2 accepts w within t steps, then M will run M_1 on w. Since $L(M_1) = L(M_2)$, M_1 will accept w. Therefore M accepts x.
  - Otherwise, M_1 does not accept w within t steps and M_2 does not accept w within t steps. Then M accepts x.

In all cases, M accepts x. Therefore $L(M) = \Sigma^*$.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string $0^t 1 w$ is not accepted by M.
    - On input $0^t 1 w$, M will find that M_1 accepts w within t steps.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string $0^t 1w$ is not accepted by M.
    - On input $0^t 1w$, M will find that M_1 accepts w within t steps.
    - M will then run M_2 on w.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string $0^t 1 w$ is not accepted by M.
  - On input $0^t 1 w$, M will find that M_1 accepts w within t steps.
  - M will then run M_2 on w.
  - Since M_2 does not accept w, M_2 will either reject or not halt.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string $0^t 1w$ is not accepted by M.
  - On input $0^t 1w$, M will find that M_1 accepts w within t steps.
  - M will then run M_2 on w.
  - Since M_2 does not accept w, M_2 will either reject or not halt.
    - If M_2 rejects w, M will reject $0^t 1w$.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string $0^t 1w$ is not accepted by M.
    - On input $0^t 1w$, M will find that M_1 accepts w within t steps.
    - M will then run M_2 on w.
    - Since M_2 does not accept w, M_2 will either reject or not halt.
        - If M_2 rejects w, M will reject $0^t 1w$.
        - If M_2 does not halt on w, M will not halt.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string 0^t1w is not accepted by M.
  - On input 0^t1w, M will find that M_1 accepts w within t steps.
  - M will then run M_2 on w.
  - Since M_2 does not accept w, M_2 will either reject or not halt.
    - If M_2 rejects w, M will reject 0^t1w.
    - If M_2 does not halt on w, M will not halt.

  In either case, M does not accept 0^t1w.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string 0^t1w is not accepted by M.
  - On input 0^t1w, M will find that M_1 accepts w within t steps.
  - M will then run M_2 on w.
  - Since M_2 does not accept w, M_2 will either reject or not halt.
    - If M_2 rejects w, M will reject 0^t1w.
    - If M_2 does not halt on w, M will not halt.

  In either case, M does not accept 0^t1w.
- Therefore $L(M) \neq \Sigma^*$.

Now suppose $L(M_1) \neq L(M_2)$. We want to show $L(M) \neq \Sigma^*$.

There is some $w \in L(M_1) \Delta L(M_2)$, where Δ denotes symmetric difference.

Assume $w \in L(M_1) - L(M_2)$.

- Let t be the number of steps used by M_1 when it accepts w.
- We claim the string $0^t 1 w$ is not accepted by M.
  - On input $0^t 1 w$, M will find that M_1 accepts w within t steps.
  - M will then run M_2 on w.
  - Since M_2 does not accept w, M_2 will either reject or not halt.
    - If M_2 rejects w, M will reject $0^t 1 w$.
    - If M_2 does not halt on w, M will not halt.

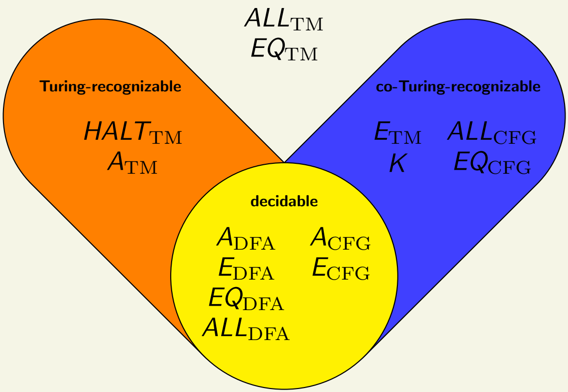
  In either case, M does not accept $0^t 1 w$.
- Therefore $L(M) \neq \Sigma^*$.

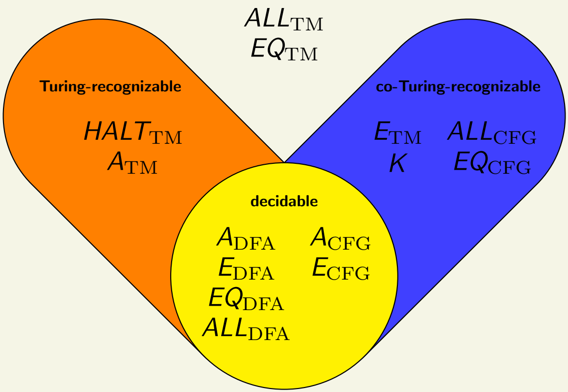
Analogoulsy, if $w \in L(M_2) - L(M_1)$, then $L(M) \neq \emptyset$.

We have shown

$$\begin{aligned} L(M_1) = L(M_2) &\quad\Rightarrow\quad L(M) = \Sigma^*, \\ L(M_1) \neq L(M_2) &\quad\Rightarrow\quad L(M) \neq \Sigma^*. \end{aligned}$$

Since the function $f(\langle M_1, M_2 \rangle) = \langle M \rangle$ is computable, this shows $EQ_{\mathrm{TM}} \leq_{\mathrm{m}} ALL_{\mathrm{TM}}$. Therefore ALL_{TM} is neither Turing-recognizable nor co-Turing-recognizable. $\square$


ALL_{TM}
EQ_{TM}
Turing-recognizable
$HALT_{\mathrm{TM}}$
A_{TM}
co-Turing-recognizable
E_{TM}
K
ALL_{CFG}
EQ_{CFG}
decidable
A_{DFA}
E_{DFA}
EQ_{DFA}
ALL_{DFA}
A_{CFG}
E_{CFG}



	DFA	CFG	TM
A (acceptance)	decidable	decidable	Turing-recognizable not co-Turing-recognizable
E (emptiness)	decidable	decidable	co-Turing-recognizable not Turing-recognizable
ALL (all strings)	decidable	co-Turing-recognizable not Turing-recognizable	not Turing-recognizable not co-Turing-recognizable
EQ (equivalence)	decidable	co-Turing-recognizable not Turing-recognizable	not Turing-recognizable not co-Turing-recognizable

Rice's Theorem

Let P be any problem about Turing machines that satisfies the following two properties. $(P \subseteq \{\langle M \rangle \mid M \text{ is a TM}\})$

Rice's Theorem

Let P be any problem about Turing machines that satisfies the following two properties. ($P \subseteq \{\langle M \rangle \mid M \text{ is a TM}\}$)

- For any two TMs M_1 and M_2, where $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P \Leftrightarrow \langle M_2 \rangle \in P$.

Rice's Theorem

Let P be any problem about Turing machines that satisfies the following two properties. ($P \subseteq \{\langle M \rangle \mid M \text{ is a TM}\}$)

- For any two TMs M_1 and M_2, where $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P \Leftrightarrow \langle M_2 \rangle \in P$. In other words, the membership of $\langle M \rangle \in P$ depends only on $L(M)$.

Rice's Theorem

Let P be any problem about Turing machines that satisfies the following two properties. $(P \subseteq \{\langle M \rangle \mid M \text{ is a TM}\})$

- For any two TMs M_1 and M_2, where $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P \Leftrightarrow \langle M_2 \rangle \in P$. In other words, the membership of $\langle M \rangle \in P$ depends only on $L(M)$.
- There exist two TMs M_1 and M_2, where $\langle M_1 \rangle \in P$ and $\langle M_2 \rangle \notin P$.

Rice's Theorem

Let P be any problem about Turing machines that satisfies the following two properties. $(P \subseteq \{\langle M \rangle \mid M \text{ is a TM}\})$

- For any two TMs M_1 and M_2, where $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P \Leftrightarrow \langle M_2 \rangle \in P$. In other words, the membership of $\langle M \rangle \in P$ depends only on $L(M)$.
- There exist two TMs M_1 and M_2, where $\langle M_1 \rangle \in P$ and $\langle M_2 \rangle \notin P$. In other words, P is nontrivial – it holds for some, but not all TMs.

Rice's Theorem

Let P be any problem about Turing machines that satisfies the following two properties. $(P \subseteq \{\langle M \rangle \mid M \text{ is a TM}\})$

- For any two TMs M_1 and M_2, where $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P \Leftrightarrow \langle M_2 \rangle \in P$. In other words, the membership of $\langle M \rangle \in P$ depends only on $L(M)$.
- There exist two TMs M_1 and M_2, where $\langle M_1 \rangle \in P$ and $\langle M_2 \rangle \notin P$. In other words, P is nontrivial – it holds for some, but not all TMs.

Then P is undecidable.

Proof. Let M_{empty} be a TM with $L(M_{\text{empty}}) = \emptyset$. We distinguish two cases.

Case 1: $\langle M_{\text{empty}} \rangle \notin P$. In this case, we show that $A_{\text{TM}} \leq_{\text{m}} P$.

Proof. Let M_{empty} be a TM with $L(M_{\text{empty}}) = \emptyset$. We distinguish two cases.

Case 1: $\langle M_{\text{empty}} \rangle \notin P$. In this case, we show that $A_{\text{TM}} \leq_{\text{m}} P$. Let N be a TM with $\langle N \rangle \in P$. Such an N exists because P is nontrival. Then since $\langle N \rangle \in P$ and $\langle M_{\text{empty}} \rangle \notin P$, we know that $L(N) \neq L(M_{\text{empty}})$, so $L(N) \neq \emptyset$.

Proof. Let M_{empty} be a TM with $L(M_{\text{empty}}) = \emptyset$. We distinguish two cases.

Case 1: $\langle M_{\text{empty}} \rangle \notin P$. In this case, we show that $A_{\text{TM}} \leq_{\text{m}} P$. Let N be a TM with $\langle N \rangle \in P$. Such an N exists because P is nontrival. Then since $\langle N \rangle \in P$ and $\langle M_{\text{empty}} \rangle \notin P$, we know that $L(N) \neq L(M_{\text{empty}})$, so $L(N) \neq \emptyset$.

For any TM M and input w, define a new TM $S_{M,w}$ as follows.

$S_{M,w}$: On input x:

1. Simulate M on input w.
2. If M rejects w, reject.
3. If M accepts w, simulate N on x and accept/reject according to what N does.

Proof. Let M_{empty} be a TM with $L(M_{\mathrm{empty}}) = \emptyset$. We distinguish two cases.

Case 1: $\langle M_{\mathrm{empty}} \rangle \notin P$. In this case, we show that $A_{\mathrm{TM}} \leq_{\mathrm{m}} P$. Let N be a TM with $\langle N \rangle \in P$. Such an N exists because P is nontrival. Then since $\langle N \rangle \in P$ and $\langle M_{\mathrm{empty}} \rangle \notin P$, we know that $L(N) \neq L(M_{\mathrm{empty}})$, so $L(N) \neq \emptyset$.

For any TM M and input w, define a new TM $S_{M,w}$ as follows.

$S_{M,w}$: On input x:

1. Simulate M on input w.
2. If M rejects w, reject.
3. If M accepts w, simulate N on x and accept/reject according to what N does.

Our reduction f is defined as

$$f(\langle M, w \rangle) = \langle S_{M,w} \rangle.$$

We have

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(S_{M,w}) = L(N) \\
&\Rightarrow \langle S_{M,w}\rangle \in P \\
&\Rightarrow f(\langle M, w\rangle) \in P
\end{aligned}$$

and

We have

$$\begin{aligned}
\langle M, w\rangle \in A_{\mathrm{TM}} &\Rightarrow L(S_{M,w}) = L(N) \\
&\Rightarrow \langle S_{M,w}\rangle \in P \\
&\Rightarrow f(\langle M, w\rangle) \in P
\end{aligned}$$

and

$$\begin{aligned}
\langle M, w\rangle \notin A_{\mathrm{TM}} &\Rightarrow L(S_{M,w}) = \emptyset \\
&\Rightarrow \langle S_{M,w}\rangle \notin P \\
&\Rightarrow f(\langle M, w\rangle) \notin P,
\end{aligned}$$

so $A_{\mathrm{TM}} \leq_{\mathrm{m}} P$ via f. Therefore P is undecidable.

Case 2: $\langle M_{\mathrm{empty}} \rangle \in P$. In this case, define $Q = P^c$. Since the decidable languages are closed under complement, it suffices to show that Q is undecidable.

Case 2: $\langle M_{\text{empty}} \rangle \in P$. In this case, define $Q = P^c$. Since the decidable languages are closed under complement, it suffices to show that Q is undecidable.

Since $\langle M_{\text{empty}} \rangle \in P$, $\langle M_{\text{empty}} \rangle \notin Q$. Then Case 1 applies to Q and we have $A_{\text{TM}} \leq_{\text{m}} Q$. Therefore Q is undecidable. $\square$

Applications of Rice's Theorem

$E_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \emptyset\}$.

Rice's Theorem applies to show that E_{TM} is undecidable:

Applications of Rice's Theorem

$E_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \emptyset\}$.

Rice's Theorem applies to show that E_{TM} is undecidable:

1. For any M_1 and M_2 with $L(M_1) = L(M_2)$, we have $L(M_1) = \emptyset \Leftrightarrow L(M_2) = \emptyset$, so $\langle M_1 \rangle \in E_{\mathrm{TM}} \Leftrightarrow \langle M_2 \rangle \in E_{\mathrm{TM}}$.

Applications of Rice's Theorem

$E_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \emptyset\}$.

Rice's Theorem applies to show that E_{TM} is undecidable:

1. For any M_1 and M_2 with $L(M_1) = L(M_2)$, we have $L(M_1) = \emptyset \Leftrightarrow L(M_2) = \emptyset$, so $\langle M_1 \rangle \in E_{\mathrm{TM}} \Leftrightarrow \langle M_2 \rangle \in E_{\mathrm{TM}}$.
2. For the second condition, let M_1 accept $\emptyset$ and M_2 accept Σ^*. Then $\langle M_1 \rangle \in E_{\mathrm{TM}}$ and $\langle M_2 \rangle \notin E_{\mathrm{TM}}$.

Applications of Rice's Theorem

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) \text{ is a regular language}\}.$

Rice's Theorem applies to show that $REGULAR_{\mathrm{TM}}$ is undecidable:

Applications of Rice's Theorem

$\mathit{REGULAR}_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) \text{ is a regular language}\}$.

Rice's Theorem applies to show that $\mathit{REGULAR}_{\mathrm{TM}}$ is undecidable:

1. The first condition is satisfied: if $L(M_1) = L(M_2)$ then $\langle M_1 \rangle \in \mathit{REGULAR}_{\mathrm{TM}} \Leftrightarrow \langle M_2 \rangle \in \mathit{REGULAR}_{\mathrm{TM}}$.

Applications of Rice's Theorem

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) \text{ is a regular language}\}$.

Rice's Theorem applies to show that $REGULAR_{\mathrm{TM}}$ is undecidable:

1. The first condition is satisfied: if $L(M_1) = L(M_2)$ then $\langle M_1 \rangle \in REGULAR_{\mathrm{TM}} \Leftrightarrow \langle M_2 \rangle \in REGULAR_{\mathrm{TM}}$.
2. For the second condition, let M_1 accept Σ^* and M_2 accept $\{0^n 1^n \mid n \geq 0\}$. Then $\langle M_1 \rangle \in REGULAR_{\mathrm{TM}}$ and $\langle M_2 \rangle \notin REGULAR_{\mathrm{TM}}$.

Applications of Rice's Theorem

$ALL_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \Sigma^*\}$.

Rice's Theorem applies to show that ALL_{TM} is undecidable:

Applications of Rice's Theorem

$ALL_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \Sigma^*\}$.

Rice's Theorem applies to show that ALL_{TM} is undecidable:

1. The first condition is satisfied: if $L(M_1) = L(M_2)$ then $\langle M_1 \rangle \in ALL_{\mathrm{TM}} \Leftrightarrow \langle M_2 \rangle \in ALL_{\mathrm{TM}}$.

Applications of Rice's Theorem

$ALL_{\mathrm{TM}} = \{\langle M \rangle \mid L(M) = \Sigma^*\}$.

Rice's Theorem applies to show that ALL_{TM} is undecidable:

1. The first condition is satisfied: if $L(M_1) = L(M_2)$ then $\langle M_1 \rangle \in ALL_{\mathrm{TM}} \Leftrightarrow \langle M_2 \rangle \in ALL_{\mathrm{TM}}$.
2. For the second condition, let M_1 accept Σ^* and M_2 accept $\emptyset$. Then $\langle M_1 \rangle \in ALL_{\mathrm{TM}}$ and $\langle M_2 \rangle \notin ALL_{\mathrm{TM}}$.