Lesson 8. Reducibility

CSIE 3110 – Formal Languages and Automata Theory

Tony Tan
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Table of contents

Table of contents

Recall

$$\begin{aligned} \mathsf{HALT} &:= \{\lfloor \mathcal{M} \rfloor \$ w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}. \\ \mathsf{HALT}_0 &:= \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ accepts } \lfloor \mathcal{M} \rfloor\}. \\ \mathsf{HALT}'_0 &:= \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ does not accept } \lfloor \mathcal{M} \rfloor\}. \end{aligned}$$

Recall

$$\begin{aligned}
\mathsf{HALT} &:= \{\lfloor\mathcal{M}\rfloor\$w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}.\\
\mathsf{HALT}_0 &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts } \lfloor\mathcal{M}\rfloor\}.\\
\mathsf{HALT}_0' &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ does not accept } \lfloor\mathcal{M}\rfloor\}.
\end{aligned}$$

In Lesson 7 we proved that HALT_0' is undecidable (by contradiction).

Recall

$$\begin{aligned}
\mathsf{HALT} &:= \{\lfloor\mathcal{M}\rfloor\$w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}.\\
\mathsf{HALT}_0 &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts } \lfloor\mathcal{M}\rfloor\}.\\
\mathsf{HALT}_0' &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ does not accept } \lfloor\mathcal{M}\rfloor\}.
\end{aligned}$$

In Lesson 7 we proved that HALT_0' is undecidable (by contradiction).

This is the only language that we proved directly to be undecidable.

Recall

$$\begin{aligned}
\mathsf{HALT} &:= \{\lfloor\mathcal{M}\rfloor\$w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}.\\
\mathsf{HALT}_0 &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts } \lfloor\mathcal{M}\rfloor\}.\\
\mathsf{HALT}_0' &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ does not accept } \lfloor\mathcal{M}\rfloor\}.
\end{aligned}$$

In Lesson 7 we proved that HALT_0' is undecidable (by contradiction).

This is the only language that we proved directly to be undecidable.

HALT_0 is undecidable because it is the "complement" of HALT_0'.

Recall

$$\begin{aligned}\mathsf{HALT} &:= \{\lfloor\mathcal{M}\rfloor\$w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}.\\ \mathsf{HALT}_0 &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts } \lfloor\mathcal{M}\rfloor\}.\\ \mathsf{HALT}'_0 &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ does not accept } \lfloor\mathcal{M}\rfloor\}.\end{aligned}$$

In Lesson 7 we proved that HALT'_0 is undecidable (by contradiction).

This is the only language that we proved directly to be undecidable.

HALT_0 is undecidable because it is the "complement" of HALT'_0.

HALT is undecidable because it is a more "general" language than HALT_0.

Recall

$$\begin{aligned} \mathsf{HALT} &:= \{\lfloor \mathcal{M} \rfloor \$ w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}. \\ \mathsf{HALT}_0 &:= \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ accepts } \lfloor \mathcal{M} \rfloor\}. \\ \mathsf{HALT}_0' &:= \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ does not accept } \lfloor \mathcal{M} \rfloor\}. \end{aligned}$$

In Lesson 7 we proved that HALT_0' is undecidable (by contradiction).

This is the only language that we proved directly to be undecidable.

HALT_0 is undecidable because it is the "complement" of HALT_0'.

HALT is undecidable because it is a more "general" language than HALT_0.

This technique is called *reductions*.

The main idea of reductions

Suppose we are given two problems (languages) K and L.

The main idea of reductions

Suppose we are given two problems (languages) K and L.

Suppose we can show how to "reduce" problem K to problem L.

The main idea of reductions

Suppose we are given two problems (languages) K and L.

Suppose we can show how to "reduce" problem K to problem L.

Intuitively, this reduction means problem L is more "general" than problem K.

The main idea of reductions

Suppose we are given two problems (languages) K and L.

Suppose we can show how to "reduce" problem K to problem L.

Intuitively, this reduction means problem L is more "general" than problem K.

That is, problem L is "harder" than problem K.

The main idea of reductions

Suppose we are given two problems (languages) K and L.

Suppose we can show how to “reduce” problem K to problem L.

Intuitively, this reduction means problem L is more “general” than problem K.

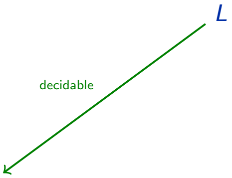
That is, problem L is “harder” than problem K.

So, if problem K is undecidable, then problem L is undecidable too.

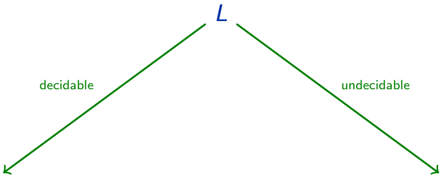
The general strategy to deal with a problem/language

L

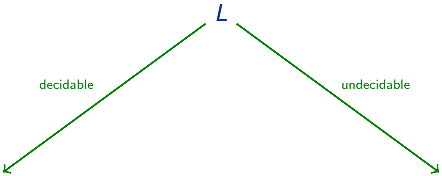
The general strategy to deal with a problem/language



The general strategy to deal with a problem/language

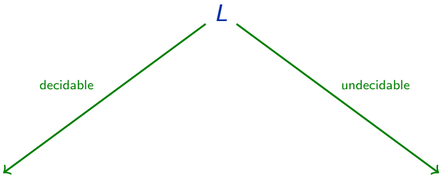


The general strategy to deal with a problem/language



- Find an algorithm for L.

The general strategy to deal with a problem/language



- Find an algorithm for L.

- Find a language K that is known to be undecidable.
- Show how to reduce K to L

The general strategy to deal with a problem/language

L

decidable

- Find an algorithm for L.

undecidable

- Find a language K that is known to be undecidable.
- Show how to reduce K to L

The focus of this lesson

The general strategy to deal with a problem/language

L

decidable

- Find an algorithm for L.

undecidable

- Find a language K that is known to be undecidable.
- Show how to reduce K to L

The focus of this lesson

Two types of reductions: *Mapping* reductions and *Turing* reductions.

Computable functions

Let $F : \Sigma^* \to \Sigma^*$ be a function from Σ^* to Σ^*.

(Def.) A Turing machine $\mathcal{M}$ computes the function F, if $\mathcal{M}$ is a 2-tape Turing machine that accepts every word $w \in \Sigma^*$ and when it halts, the content of its second tape is $F(w)$.

Computable functions

Let $F : \Sigma^* \to \Sigma^*$ be a function from Σ^* to Σ^*.

(Def.) A Turing machine $\mathcal{M}$ computes the function F, if $\mathcal{M}$ is a 2-tape Turing machine that accepts every word $w \in \Sigma^*$ and when it halts, the content of its second tape is $F(w)$.

Note that for $\mathcal{M}$ to compute F, the content of the first tape can be anything when it halts. The main point is that when $\mathcal{M}$ halts, the content of the second tape is $F(w)$.

Computable functions

Let $F : \Sigma^* \to \Sigma^*$ be a function from Σ^* to Σ^*.

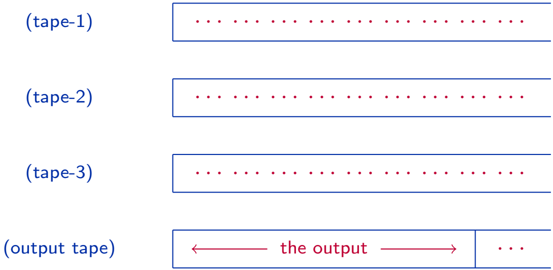
(Def.) A Turing machine $\mathcal{M}$ computes the function F, if $\mathcal{M}$ is a 2-tape Turing machine that accepts every word $w \in \Sigma^*$ and when it halts, the content of its second tape is $F(w)$.

Note that for $\mathcal{M}$ to compute F, the content of the first tape can be anything when it halts. The main point is that when $\mathcal{M}$ halts, the content of the second tape is $F(w)$.

(Def.) A function $F : \Sigma^* \to \Sigma^*$ is *computable*, if there is a Turing machine that computes it.

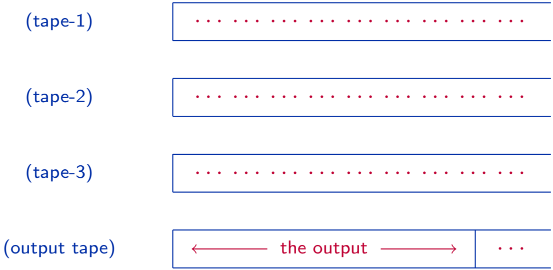
Computable functions by multi-tape Turing machines

The Turing machine $\mathcal{M}$ that computes F can be any multi-tape Turing machine with a designated output tape that contains the output string.



Computable functions by multi-tape Turing machines

The Turing machine $\mathcal{M}$ that computes F can be any multi-tape Turing machine with a designated output tape that contains the output string.



(Note) Any function that can be computed by a multi-tape Turing machine can also be computed by a 2-tape Turing machine.

Mapping reductions

(Def.) A language L_1 is *mapping reducible* to another language L_2, denoted by:

$$L_1 \;\leqslant_m\; L_2$$

if there is a computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

Mapping reductions

(Def.) A language L_1 is *mapping reducible* to another language L_2, denoted by:

$$L_1 \leqslant_m L_2$$

if there is a computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

The function F is called *mapping reduction*.

Mapping reductions

(Def.) A language L_1 is *mapping reducible* to another language L_2, denoted by:

$$L_1 \ \leqslant_m \ L_2$$

if there is a computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

The function F is called *mapping reduction*.

Intuitively $L_1 \leqslant_m L_2$ means "L_2 is (computationally) more general than L_1".

Mapping reductions

(Def.) A language L_1 is *mapping reducible* to another language L_2, denoted by:

$$L_1 \ \leqslant_m \ L_2$$

if there is a computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

The function F is called *mapping reduction*.

Intuitively $L_1 \leqslant_m L_2$ means "L_2 is (computationally) more general than L_1".

It also means that a Turing machine that decides L_2 can be used to decide L_1.

Turing reductions

(Def.) A language L_1 is *Turing reducible* to another language L_2, denoted by:

$$L_1 \leqslant_T L_2$$

if there is a Turing machine $\mathcal{M}_2$ that decides L_2, then there is a Turing machine $\mathcal{M}_1$ that decides L_1 using $\mathcal{M}_2$ as a "subroutine."

Turing reductions

(Def.) A language L_1 is *Turing reducible* to another language L_2, denoted by:

$$L_1 \leqslant_T L_2$$

if there is a Turing machine $\mathcal{M}_2$ that decides L_2, then there is a Turing machine $\mathcal{M}_1$ that decides L_1 using $\mathcal{M}_2$ as a "subroutine."

Here we assume that $\mathcal{M}_2$ decides L_2 in *one* step.

Turing reductions

(Def.) A language L_1 is *Turing reducible* to another language L_2, denoted by:

$$L_1 \leqslant_T L_2$$

if there is a Turing machine $\mathcal{M}_2$ that decides L_2, then there is a Turing machine $\mathcal{M}_1$ that decides L_1 using $\mathcal{M}_2$ as a "subroutine."

Here we assume that $\mathcal{M}_2$ decides L_2 in *one* step.

(Def.) We call $\mathcal{M}_1$ a Turing machine *with oracle access to L_2*.

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

$(L_1 \leqslant_m L_2)$

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

($L_1 \leqslant_m L_2$)

On input w:

{ $\cdots\cdots$

$\cdots\cdots$

$\vdots$

$\vdots$ *some algorithm*

Output v

}

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

($L_1 \leqslant_m L_2$)

$$\begin{array}{l} \textit{On input } w\textit{:} \\ \{ \quad \left.\begin{array}{l} \cdots\cdots \\ \cdots\cdots \\ \quad\vdots \\ \quad\vdots \end{array}\right\} \textit{some algorithm} \\ \quad \textit{Output } v \\ \} \end{array}$$

- $w \in L_1$ if and only if $v \in L_2$.

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

$(L_1 \leqslant_m L_2)$

On input w:

$\{$ ......

......

⋮

⋮ $\Bigg\}$ *some algorithm*

Output v

$\}$

- $w \in L_1$ if and only if $v \in L_2$. $\Rightarrow$ Very important!

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

($L_1 \leqslant_m L_2$)

On input w:

$$\left.\begin{array}{l} \{ \quad \cdots\cdots \\ \quad\quad \cdots\cdots \\ \quad\quad\quad \vdots \\ \quad\quad\quad \vdots \end{array}\right\} \textit{some algorithm}$$

Output v

}

- $w \in L_1$ if and only if $v \in L_2$. $\Rightarrow$ Very important!
- Inside the algorithm we do not assume/use anything about L_2.

Mapping reductions vs. Turing reductions

On the surface, mapping reductions and Turing reductions look similar, but they are different.

$(L_1 \leqslant_m L_2)$

On input w:
{ $\cdots\cdots$
$\cdots\cdots$
$\vdots$
$\vdots$
Output v
}

some algorithm

- $w \in L_1$ if and only if $v \in L_2$. $\Rightarrow$ Very important!
- Inside the algorithm we do not assume/use anything about L_2.
- View it this way: If L_2 is decidable by, say, $\mathcal{M}_2$, then in the algorithm we can only use $\mathcal{M}_2$ once(!).
  The answer provided by $\mathcal{M}_2$ must also be the answer to whether $w \in L_1$.

Mapping reductions vs. Turing reductions – continued

($L_1 \leqslant_T L_2$)

Mapping reductions vs. Turing reductions – continued

$(L_1 \leqslant_T L_2)$

$$\left.\begin{array}{l} \textit{On input } w\textit{:} \\ \{ \quad \cdots\cdots \\ \quad\quad \cdots\cdots \\ \quad\quad\quad \vdots \\ \quad\quad\quad \vdots \\ \quad\quad\quad \vdots \\ \} \end{array}\right\} \textit{some algorithm that decides } L_1$$

Mapping reductions vs. Turing reductions – continued

$(L_1 \leqslant_T L_2)$

$$\left.\begin{array}{l} \textit{On input } w\text{:} \\ \{ \quad \cdots\cdots \\ \quad\;\; \cdots\cdots \\ \qquad \vdots \\ \qquad \vdots \\ \qquad \vdots \\ \;\} \end{array}\right\} \textit{some algorithm that decides } L_1$$

We assume a Turing machine $\mathcal{M}_2$ that decides L_2.

Mapping reductions vs. Turing reductions – continued

($L_1 \leqslant_T L_2$)

$$\begin{array}{l} \textit{On input } w\text{:} \\ \left\{ \quad \left.\begin{array}{c} \cdots\cdots \\ \cdots\cdots \\ \vdots \\ \vdots \\ \vdots \end{array}\right\} \textit{some algorithm that decides } L_1 \right. \\ \} \end{array}$$

We assume a Turing machine $\mathcal{M}_2$ that decides L_2.

- Inside the algorithm the Turing machine $\mathcal{M}_2$ can be called multiple times.

Mapping reductions vs. Turing reductions – continued

($L_1 \leqslant_T L_2$)

On input w:

$$\left.\begin{array}{l} \{ \quad \cdots\cdots \\ \quad\;\; \cdots\cdots \\ \quad\quad \vdots \\ \quad\quad \vdots \\ \quad\quad \vdots \\ \} \end{array}\right\} \textit{some algorithm that decides } L_1$$

We assume a Turing machine $\mathcal{M}_2$ that decides L_2.

- Inside the algorithm the Turing machine $\mathcal{M}_2$ can be called multiple times.
- The (multiple) answers provided by $\mathcal{M}_2$ can be used to decided whether $w \in L_1$.

Example of a mapping reduction

$\mathsf{HALT}_0 \leqslant_m \mathsf{HALT}$ via the following reduction:

Example of a mapping reduction

$\mathsf{HALT}_0 \leqslant_m \mathsf{HALT}$ via the following reduction:

On input $\lfloor \mathcal{M} \rfloor$*:*
{ *Output* $\lfloor \mathcal{M} \rfloor \$ \lfloor \mathcal{M} \rfloor$.
}

Example of a mapping reduction

$\mathsf{HALT}_0 \leqslant_m \mathsf{HALT}$ via the following reduction:

On input $\lfloor\mathcal{M}\rfloor$*:*
$\{$ *Output* $\lfloor\mathcal{M}\rfloor\$\lfloor\mathcal{M}\rfloor$.
$\}$

Note that:

$$\lfloor\mathcal{M}\rfloor \in \mathsf{HALT}_0 \qquad \text{if and only if} \qquad \lfloor\mathcal{M}\rfloor\$\lfloor\mathcal{M}\rfloor \in \mathsf{HALT}$$

Example of a Turing reduction

$\mathsf{HALT}_0' \leqslant_T \mathsf{HALT}_0$ via the following reduction:

Example of a Turing reduction

$\mathrm{HALT}_0' \leqslant_T \mathrm{HALT}_0$ via the following reduction:

We assume that there is Turing machine $\mathcal{A}$ that decides HALT_0.

On input $\lfloor\mathcal{M}\rfloor$*:*
{ *Run* $\mathcal{A}$ *on* $\lfloor\mathcal{M}\rfloor$.
If ($\mathcal{A}$ *accepts* $\lfloor\mathcal{M}\rfloor$)
REJECT.
else
ACCEPT.
}

Example of a Turing reduction

$\mathrm{HALT}_0' \leqslant_T \mathrm{HALT}_0$ via the following reduction:

We assume that there is Turing machine $\mathcal{A}$ that decides HALT_0.

On input $\lfloor \mathcal{M} \rfloor$*:*
$\{$ *Run* $\mathcal{A}$ *on* $\lfloor \mathcal{M} \rfloor$*.*
If ($\mathcal{A}$ *accepts* $\lfloor \mathcal{M} \rfloor$)
REJECT.
else
ACCEPT.
$\}$

In this algorithm we call $\mathcal{A}$ only once, but it makes some change to the answer it provides.

Example of a Turing reduction

$\mathsf{HALT}_0' \leqslant_T \mathsf{HALT}_0$ via the following reduction:

We assume that there is Turing machine $\mathcal{A}$ that decides HALT_0.

On input $\lfloor \mathcal{M} \rfloor$*:*
{ *Run* $\mathcal{A}$ *on* $\lfloor \mathcal{M} \rfloor$.
If ($\mathcal{A}$ *accepts* $\lfloor \mathcal{M} \rfloor$)
REJECT.
else
ACCEPT.
}

In this algorithm we call $\mathcal{A}$ only once, but it makes some change to the answer it provides.

- If the answer from $\mathcal{A}$ is "accept", the algorithm "rejects".

Example of a Turing reduction

$\mathrm{HALT}_0' \leqslant_T \mathrm{HALT}_0$ via the following reduction:

We assume that there is Turing machine $\mathcal{A}$ that decides HALT_0.

On input $\lfloor \mathcal{M} \rfloor$*:*
{ *Run* $\mathcal{A}$ *on* $\lfloor \mathcal{M} \rfloor$*.*
If ($\mathcal{A}$ *accepts* $\lfloor \mathcal{M} \rfloor$)
REJECT.
else
ACCEPT.
}

In this algorithm we call $\mathcal{A}$ only once, but it makes some change to the answer it provides.

- If the answer from $\mathcal{A}$ is "accept", the algorithm "rejects".
- If the answer from $\mathcal{A}$ is "reject", the algorithm "accepts".

Some observations

Some observations

- If $L_1 \leqslant_m L_2$, then $L_1 \leqslant_T L_2$.

Some observations

- If $L_1 \leqslant_m L_2$, then $L_1 \leqslant_T L_2$.
- If $L_1 \leqslant_T L_2$ and L_1 is undecidable, then L_2 is also undecidable.

Some observations

- If $L_1 \leqslant_m L_2$, then $L_1 \leqslant_T L_2$.
- If $L_1 \leqslant_T L_2$ and L_1 is undecidable, then L_2 is also undecidable.

(Important) The following is NOT true.

Some observations

- If $L_1 \leqslant_m L_2$, then $L_1 \leqslant_T L_2$.
- If $L_1 \leqslant_T L_2$ and L_1 is undecidable, then L_2 is also undecidable.

(Important) The following is NOT true.

- If $L_1 \leqslant_T L_2$ and L_2 is undecidable, then L_1 is undecidable.

Some observations

- If $L_1 \leqslant_m L_2$, then $L_1 \leqslant_T L_2$.
- If $L_1 \leqslant_T L_2$ and L_1 is undecidable, then L_2 is also undecidable.

(Important) The following is NOT true.

- If $L_1 \leqslant_T L_2$ and L_2 is undecidable, then L_1 is undecidable.
- If $L_1 \leqslant_m L_2$ and L_2 is undecidable, then L_1 is undecidable.

Table of contents

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.
- $L_1 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{0, 1\}^*\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_1$ if and only if $\mathcal{M}$ accepts every word.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.
- $L_1 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{0,1\}^*\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_1$ if and only if $\mathcal{M}$ accepts every word.
- $L_2 := \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ accepts the empty word } \varepsilon\}$
  That is, $\lfloor \mathcal{M} \rfloor \in L_2$ if and only if $\mathcal{M}$ accepts the empty word ε.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor\mathcal{M}\rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.
- $L_1 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \{0,1\}^*\}$.
  That is, $\lfloor\mathcal{M}\rfloor \in L_1$ if and only if $\mathcal{M}$ accepts every word.
- $L_2 := \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts the empty word } \varepsilon\}$
  That is, $\lfloor\mathcal{M}\rfloor \in L_2$ if and only if $\mathcal{M}$ accepts the empty word ε.
- $L_3 := \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts the word } 1101\}$.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor\mathcal{M}\rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.
- $L_1 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \{0,1\}^*\}$.
  That is, $\lfloor\mathcal{M}\rfloor \in L_1$ if and only if $\mathcal{M}$ accepts every word.
- $L_2 := \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts the empty word } \varepsilon\}$
  That is, $\lfloor\mathcal{M}\rfloor \in L_2$ if and only if $\mathcal{M}$ accepts the empty word ε.
- $L_3 := \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts the word } 1101\}$.
- $L_4 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor \mathcal{M} \rfloor \ \mid \ L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.
- $L_1 := \{\lfloor \mathcal{M} \rfloor \ \mid \ L(\mathcal{M}) = \{0, 1\}^*\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_1$ if and only if $\mathcal{M}$ accepts every word.
- $L_2 := \{\lfloor \mathcal{M} \rfloor \ \mid \ \mathcal{M} \text{ accepts the empty word } \varepsilon\}$
  That is, $\lfloor \mathcal{M} \rfloor \in L_2$ if and only if $\mathcal{M}$ accepts the empty word ε.
- $L_3 := \{\lfloor \mathcal{M} \rfloor \ \mid \ \mathcal{M} \text{ accepts the word } 1101\}$.
- $L_4 := \{\lfloor \mathcal{M} \rfloor \ \mid \ L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$.
- $L_5 := \{\lfloor \mathcal{M} \rfloor \ \mid \ L(\mathcal{M}) \text{ is a regular language}\}$.

Some variants of the halting problem

(Def.) $L(\mathcal{M})$ denotes the set of all words accepted by the Turing machine $\mathcal{M}$.

The following languages are all undecidable.

- $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$.
  That is, $\lfloor \mathcal{M} \rfloor \in L_0$ if and only if $\mathcal{M}$ does not accept any word.

- $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$.

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

Proof that $L_0 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor\mathcal{M}\rfloor\$w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)

- Output $\lfloor\mathcal{K}_{\mathcal{M},w}\rfloor$.

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)

- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT,

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)

- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \Sigma^*$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$.

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \Sigma^*$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT,

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)

- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \Sigma^*$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \emptyset$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \notin \overline{L}_0$.

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)

- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \Sigma^*$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \emptyset$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \notin \overline{L}_0$.

Thus,

$\lfloor \mathcal{M} \rfloor \$ w \in$ HALT if and only if $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable

We show that HALT $\leqslant_m \overline{L}_0$, where $\overline{L}_0$ is the complement of L_0.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

  (Note: ACCEPT and REJECT above are inside $\mathcal{K}_{\mathcal{M},w}$.)

- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \Sigma^*$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \emptyset$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \notin \overline{L}_0$.

Thus,

$$\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT} \quad \text{if and only if} \quad \lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in \overline{L}_0$$

So, HALT $\leqslant_m \overline{L}_0$.

Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

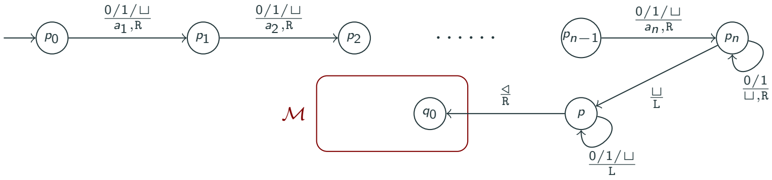
Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

Add the following: (where $w = a_1 a_2 \cdots a_n$)



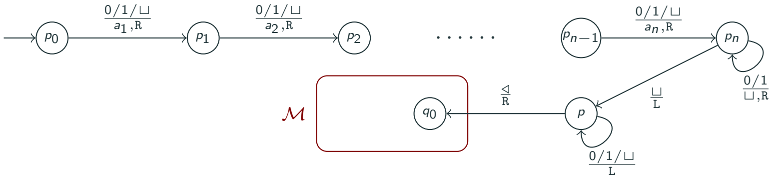
Proof that $L_0 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor\mathcal{M}\rfloor\$w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

Add the following: (where $w = a_1 a_2 \cdots a_n$)



- Make p_0 the initial state of $\mathcal{K}_{\mathcal{M},w}$.
- The accept state of $\mathcal{K}_{\mathcal{M},w}$ is the accept state of $\mathcal{M}$.
- The reject state of $\mathcal{K}_{\mathcal{M},w}$ is the reject state of $\mathcal{M}$.

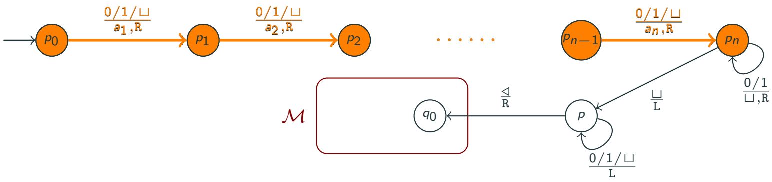
Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

Add the following: (where $w = a_1 a_2 \cdots a_n$)



Rewrite the content of the tape to be w.

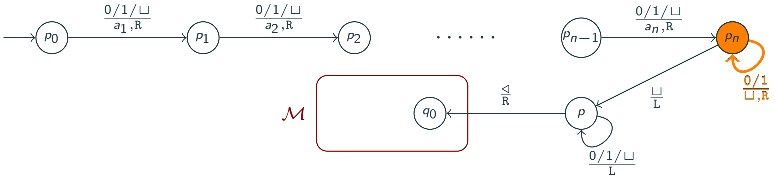
Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

Add the following: (where $w = a_1 a_2 \cdots a_n$)



"Erase" the remaining of the input v when $|v| > |w|$.

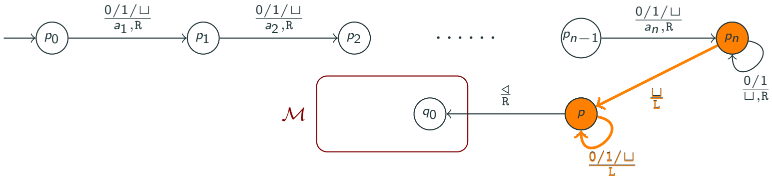
Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

Add the following: (where $w = a_1 a_2 \cdots a_n$)



Move the head back to the beginning of the tape.

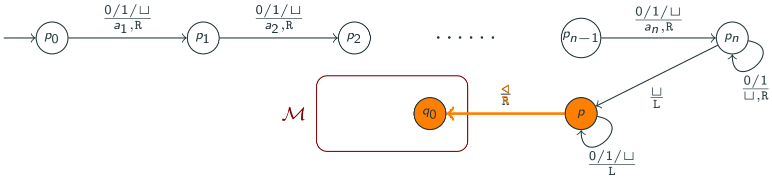
Proof that $L_0 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \emptyset\}$ is undecidable – illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  – Run $\mathcal{M}$ on w.
  – If $\mathcal{M}$ accepts w, ACCEPT.
  – If $\mathcal{M}$ rejects w, REJECT.

Add the following: (where $w = a_1 a_2 \cdots a_n$)



When the head reaches the left-end marker $\triangleleft$, it moves right.

It enters the state q_0 of $\mathcal{M}$ (i.e., to run $\mathcal{M}$ on w).

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    * Run $\mathcal{M}$ on w.
    * If $\mathcal{M}$ accepts w, ACCEPT.
    * If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \ \mid \ L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    * Run $\mathcal{M}$ on w.
    * If $\mathcal{M}$ accepts w, ACCEPT.
    * If $\mathcal{M}$ rejects w, REJECT.

- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT,

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \{a^n b^n | n \geqslant 0\}$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$.

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \{a^n b^n | n \geqslant 0\}$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT,

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \{a^n b^n | n \geqslant 0\}$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \emptyset$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \notin L_4$.

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - $*$ Run $\mathcal{M}$ on w.
    - $*$ If $\mathcal{M}$ accepts w, ACCEPT.
    - $*$ If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \{a^n b^n | n \geqslant 0\}$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \emptyset$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \notin L_4$.

Thus,

$$\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT} \quad \text{if and only if} \quad \lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$$

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable

Let $\mathcal{A}$ be a TM that decides the language $\{a^n b^n | n \geqslant 0\}$.

We show that HALT $\leqslant_m L_4$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

If $\lfloor \mathcal{M} \rfloor \$ w \in$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \{a^n b^n | n \geqslant 0\}$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{K}_{\mathcal{M},w}) = \emptyset$, so, $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \notin L_4$.

Thus,

$$\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT} \quad \text{if and only if} \quad \lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in L_4$$

So, HALT $\leqslant_m L_4$.

Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    * Run $\mathcal{M}$ on w.
    * If $\mathcal{M}$ accepts w, ACCEPT.
    * If $\mathcal{M}$ rejects w, REJECT.

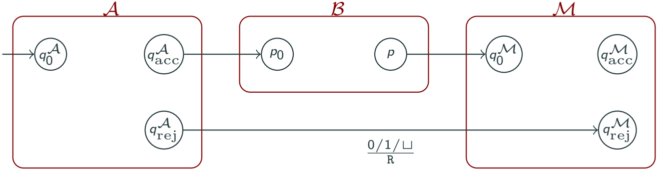
Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - ∗ Run $\mathcal{M}$ on w.
    - ∗ If $\mathcal{M}$ accepts w, ACCEPT.
    - ∗ If $\mathcal{M}$ rejects w, REJECT.

$\mathcal{K}_{\mathcal{M},w}$:



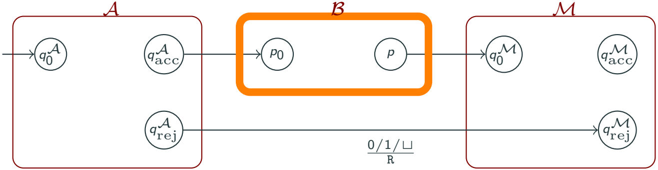
Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - $*$ Run $\mathcal{M}$ on w.
    - $*$ If $\mathcal{M}$ accepts w, ACCEPT.
    - $*$ If $\mathcal{M}$ rejects w, REJECT.

$\mathcal{K}_{\mathcal{M},w}$:



Turing machine $\mathcal{B}$ writes w on the tape and enters $q_0^{\mathcal{M}}$ (to run $\mathcal{M}$ on w).

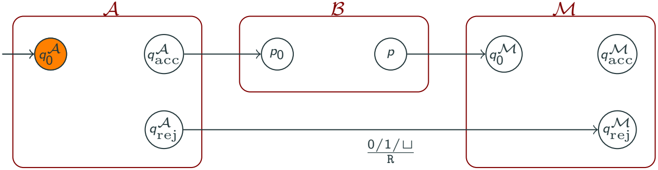
Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - ∗ Run $\mathcal{M}$ on w.
    - ∗ If $\mathcal{M}$ accepts w, ACCEPT.
    - ∗ If $\mathcal{M}$ rejects w, REJECT.

$\mathcal{K}_{\mathcal{M},w}$:



$q_0^{\mathcal{A}}$ is the initial state.

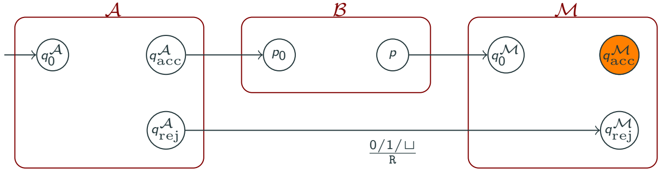
Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - ∗ Run $\mathcal{M}$ on w.
    - ∗ If $\mathcal{M}$ accepts w, ACCEPT.
    - ∗ If $\mathcal{M}$ rejects w, REJECT.

$\mathcal{K}_{\mathcal{M},w}$:



$q_{\mathrm{acc}}^{\mathcal{M}}$ is the accept state.

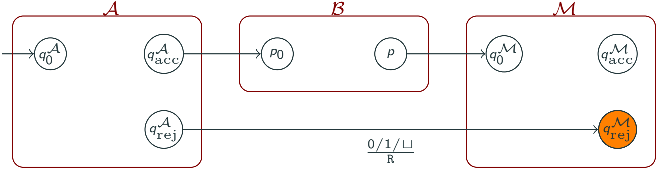
Proof that $L_4 := \{\lfloor\mathcal{M}\rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor\mathcal{M}\rfloor\$w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - ∗ Run $\mathcal{M}$ on w.
    - ∗ If $\mathcal{M}$ accepts w, ACCEPT.
    - ∗ If $\mathcal{M}$ rejects w, REJECT.

$\mathcal{K}_{\mathcal{M},w}$:



$q_{\mathrm{rej}}^{\mathcal{M}}$ is the reject state

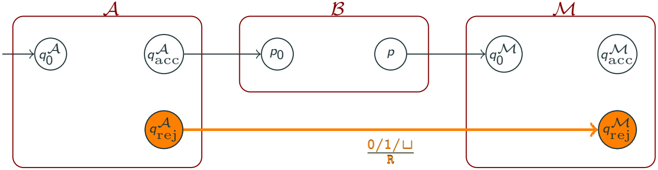
Proof that $L_4 := \{\lfloor \mathcal{M} \rfloor \mid L(\mathcal{M}) = \{a^n b^n \mid n \geqslant 0\}\}$ is undecidable – Illustration

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in \{a^n b^n | n \geqslant 0\}$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - ∗ Run $\mathcal{M}$ on w.
    - ∗ If $\mathcal{M}$ accepts w, ACCEPT.
    - ∗ If $\mathcal{M}$ rejects w, REJECT.

$\mathcal{K}_{\mathcal{M},w}$:



Add a transition so that from $q_{\mathrm{rej}}^{\mathcal{A}}$ the TM enters $q_{\mathrm{rej}}^{\mathcal{M}}$.

Rice's theorem

The proof can be generalized to the so called *Rice's theorem*.

Rice's theorem

The proof can be generalized to the so called *Rice's theorem*.

(Def.) Let P be a set of descriptions of Turing machines.

P is a *property*, if for every Turing machines $\mathcal{M}_1$ and $\mathcal{M}_2$, if:

$$L(\mathcal{M}_1) \; = \; L(\mathcal{M}_2)$$

then:

$$\text{either} \qquad \lfloor\mathcal{M}_1\rfloor, \lfloor\mathcal{M}_2\rfloor \; \in P \qquad \text{or} \qquad \lfloor\mathcal{M}_1\rfloor, \lfloor\mathcal{M}_2\rfloor \; \notin P$$

Rice's theorem

The proof can be generalized to the so called *Rice's theorem*.

(Def.) Let P be a set of descriptions of Turing machines.

P is a *property*, if for every Turing machines $\mathcal{M}_1$ and $\mathcal{M}_2$, if:

$$L(\mathcal{M}_1) = L(\mathcal{M}_2)$$

then:

$$\text{either} \qquad \lfloor\mathcal{M}_1\rfloor, \lfloor\mathcal{M}_2\rfloor \in P \qquad \text{or} \qquad \lfloor\mathcal{M}_1\rfloor, \lfloor\mathcal{M}_2\rfloor \notin P$$

The criteria for $\lfloor\mathcal{M}\rfloor$ to be in P depends on the language $L(\mathcal{M})$, and *not* on the string $\lfloor\mathcal{M}\rfloor$ itself.

Rice's theorem

The proof can be generalized to the so called *Rice's theorem*.

(Def.) Let P be a set of descriptions of Turing machines.

P is a *property*, if for every Turing machines $\mathcal{M}_1$ and $\mathcal{M}_2$, if:

$$L(\mathcal{M}_1) \ = \ L(\mathcal{M}_2)$$

then:

$$\text{either} \qquad \lfloor \mathcal{M}_1 \rfloor, \lfloor \mathcal{M}_2 \rfloor \ \in P \qquad \text{or} \qquad \lfloor \mathcal{M}_1 \rfloor, \lfloor \mathcal{M}_2 \rfloor \ \notin P$$

The criteria for $\lfloor \mathcal{M} \rfloor$ to be in P depends on the language $L(\mathcal{M})$, and *not* on the string $\lfloor \mathcal{M} \rfloor$ itself.

(Def.) A property P is called a *trivial* property, if:

either $\quad P = \emptyset \quad$ or $\quad P$ contains all the descriptions of Turing machines

Rice's theorem – continued

Theorem 8.6 (Rice's theorem)
For a property P, if P is not a trivial property, then P is undecidable.

Rice's theorem – continued

Theorem 8.6 (Rice's theorem)
For a property P, if P is not a trivial property, then P is undecidable.

(Proof) Let P be a non-trivial property.

Rice's theorem – continued

Theorem 8.6 (Rice's theorem)
For a property P, if P is not a trivial property, then P is undecidable.

(Proof) Let P be a non-trivial property.

First, we consider the case where P does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Theorem 8.6 (Rice's theorem)
For a property P, if P is not a trivial property, then P is undecidable.

(Proof) Let P be a non-trivial property.

First, we consider the case where P does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Let $\mathcal{A}$ be a Turing machine where $\lfloor \mathcal{A} \rfloor \in P$.

Rice's theorem – continued

Theorem 8.6 (Rice's theorem)
For a property P, if P is not a trivial property, then P is undecidable.

(Proof) Let P be a non-trivial property.

First, we consider the case where P does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Let $\mathcal{A}$ be a Turing machine where $\lfloor \mathcal{A} \rfloor \in P$.

Such $\mathcal{A}$ exists since P is not trivial.

The proof of Rice's theorem

We show that $\mathsf{HALT} \leqslant_m P$.

The proof of Rice's theorem

We show that HALT $\leqslant_m P$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in L(\mathcal{A})$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

The proof of Rice's theorem

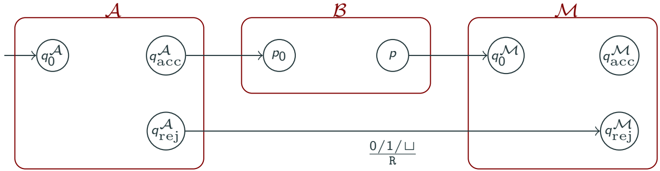
We show that HALT $\leqslant_m P$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in L(\mathcal{A})$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

$\mathcal{K}_{\mathcal{M},w}$:



The proof of Rice's theorem

We show that HALT $\leqslant_m P$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in L(\mathcal{A})$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

The proof of Rice's theorem

We show that HALT $\leqslant_m P$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in L(\mathcal{A})$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

By similar reasoning as the proof of the undecidability of L_4:

$$\mathcal{M}\$w \in \text{HALT} \quad \text{if and only if} \quad \lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in P$$

The proof of Rice's theorem

We show that HALT $\leqslant_m P$.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct the following Turing machine denoted by $\mathcal{K}_{\mathcal{M},w}$:

  On input u:
  - Run $\mathcal{A}$ on u. (to check if $u \in L(\mathcal{A})$.)
  - If $\mathcal{A}$ rejects u, REJECT.
  - If $\mathcal{A}$ accepts u:
    - Run $\mathcal{M}$ on w.
    - If $\mathcal{M}$ accepts w, ACCEPT.
    - If $\mathcal{M}$ rejects w, REJECT.
- Output $\lfloor \mathcal{K}_{\mathcal{M},w} \rfloor$.

By similar reasoning as the proof of the undecidability of L_4:

$$\mathcal{M}\$w \in \text{HALT} \quad \text{if and only if} \quad \lfloor \mathcal{K}_{\mathcal{M},w} \rfloor \in P$$

Thus, we have proved Rice's theorem for the case where P does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Consider the complement of P, denoted by $\overline{P}$.

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Consider the complement of P, denoted by $\overline{P}$.

Now $\overline{P}$ does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Consider the complement of P, denoted by $\overline{P}$.

Now $\overline{P}$ does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Since P is not a trivial property, we have $\overline{P} \neq \emptyset$.

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Consider the complement of P, denoted by $\overline{P}$.

Now $\overline{P}$ does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Since P is not a trivial property, we have $\overline{P} \neq \emptyset$.

So we can pick a Turing machine $\mathcal{A}$ where $\lfloor \mathcal{A} \rfloor \in \overline{P}$.

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Consider the complement of P, denoted by $\overline{P}$.

Now $\overline{P}$ does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Since P is not a trivial property, we have $\overline{P} \neq \emptyset$.

So we can pick a Turing machine $\mathcal{A}$ where $\lfloor \mathcal{A} \rfloor \in \overline{P}$.

The previous case already establishes $\mathsf{HALT} \leqslant_m \overline{P}$.

The proof of Rice's theorem – continued

Now we consider the case where P contains $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Consider the complement of P, denoted by $\overline{P}$.

Now $\overline{P}$ does not contain $\lfloor \mathcal{M} \rfloor$ where $L(\mathcal{M}) = \emptyset$.

Since P is not a trivial property, we have $\overline{P} \neq \emptyset$.

So we can pick a Turing machine $\mathcal{A}$ where $\lfloor \mathcal{A} \rfloor \in \overline{P}$.

The previous case already establishes $\mathsf{HALT} \leqslant_m \overline{P}$.

This means $\overline{P}$ is undecidable, and hence, P is also undecidable.

Table of contents

CFL intersection

CFL-Intersection

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$. Otherwise, output `False`.

CFL intersection

CFL-Intersection

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$. Otherwise, output `False`.

This problem can be viewed as a language:

$$\text{CFL-Intersection} := \{\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor \mid L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset\}$$

where $\lfloor \mathcal{G} \rfloor$ denotes the encoding of $\mathcal{G}$ as a string over some fixed alphabet.

CFL intersection

CFL-Intersection

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$. Otherwise, output `False`.

This problem can be viewed as a language:

$$\text{CFL-Intersection} \;:=\; \{\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor \mid L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset\}$$

where $\lfloor \mathcal{G} \rfloor$ denotes the encoding of $\mathcal{G}$ as a string over some fixed alphabet.

A CFG over Σ can be encoded using the alphabet $\Sigma \cup \{0, 1, \langle, \rangle, \rightarrow, \diamond, \#\}$.

CFL intersection

CFL-Intersection

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$. Otherwise, output `False`.

This problem can be viewed as a language:

$$\text{CFL-Intersection} := \{\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor \mid L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset\}$$

where $\lfloor \mathcal{G} \rfloor$ denotes the encoding of $\mathcal{G}$ as a string over some fixed alphabet.

A CFG over Σ can be encoded using the alphabet $\Sigma \cup \{0, 1, \langle, \rangle, \rightarrow, \diamond, \#\}$.

Let $\mathcal{G}$ be a CFG with n variables.

CFL intersection

CFL-Intersection

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$. Otherwise, output `False`.

This problem can be viewed as a language:

$$\text{CFL-Intersection} \;:=\; \{\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor \mid L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset\}$$

where $\lfloor \mathcal{G} \rfloor$ denotes the encoding of $\mathcal{G}$ as a string over some fixed alphabet.

A CFG over Σ can be encoded using the alphabet $\Sigma \cup \{0, 1, \langle, \rangle, \rightarrow, \diamond, \#\}$.

Let $\mathcal{G}$ be a CFG with n variables.

- The variables can be encoded as $\langle i \rangle$, where i is an integer (written in binary) between 0 and $n - 1$.

CFL intersection

CFL-Intersection

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$. Otherwise, output `False`.

This problem can be viewed as a language:

$$\text{CFL-Intersection} \ := \ \{\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor \mid L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset\}$$

where $\lfloor \mathcal{G} \rfloor$ denotes the encoding of $\mathcal{G}$ as a string over some fixed alphabet.

A CFG over Σ can be encoded using the alphabet $\Sigma \cup \{0, 1, \langle, \rangle, \rightarrow, \diamond, \#\}$.

Let $\mathcal{G}$ be a CFG with n variables.

- The variables can be encoded as $\langle i \rangle$, where i is an integer (written in binary) between 0 and $n-1$.
- A rule, say, $S \rightarrow 0X11$ is encoded as $\langle 0 \rangle \ \rightarrow \ 0\langle 3 \rangle 11$.
  (Assuming that S is represented as 0 and X as 3).

The problem/language CFL-Intersection is undecidable

Theorem 8.8

The problem CFL-Intersection *is undecidable.*

The problem/language CFL-Intersection is undecidable

Theorem 8.8

The problem CFL-Intersection *is undecidable.*

We will show that HALT $\leqslant_m$ CFL-Intersection.

The problem/language CFL-Intersection is undecidable

Theorem 8.8
The problem CFL-Intersection *is undecidable.*

We will show that HALT $\leqslant_m$ CFL-Intersection.

We assume that HALT contains only $\lfloor \mathcal{M} \rfloor \$ w$ where $\mathcal{M}$ is a 1-tape Turing machine and $\mathcal{M}$ accepts w.

Some observations

Let $\mathcal{M}$ be a Turing machine.

Some observations

Let $\mathcal{M}$ be a Turing machine.

- Add a "new" state q_{loop} such that instead of entering the q_{rej}, $\mathcal{M}$ enters q_{loop} and loops forever.

Some observations

Let $\mathcal{M}$ be a Turing machine.

- Add a "new" state q_{loop} such that instead of entering the q_{rej}, $\mathcal{M}$ enters q_{loop} and loops forever.
- Add some states, so that for every word w accepted by $\mathcal{M}$, the run has odd length:

  $$C_0 \vdash C_1 \vdash C_2 \vdash C_3 \vdash \cdots \vdash C_n$$

  where n is odd.

Some observations

Let $\mathcal{M}$ be a Turing machine.

- Add a "new" state q_{loop} such that instead of entering the q_{rej}, $\mathcal{M}$ enters q_{loop} and loops forever.
- Add some states, so that for every word w accepted by $\mathcal{M}$, the run has odd length:

$$C_0 \vdash C_1 \vdash C_2 \vdash C_3 \vdash \cdots \vdash C_n$$

  where n is odd.

After adding those states, the following holds for every word w:

Some observations

Let $\mathcal{M}$ be a Turing machine.

- Add a "new" state q_{loop} such that instead of entering the q_{rej}, $\mathcal{M}$ enters q_{loop} and loops forever.
- Add some states, so that for every word w accepted by $\mathcal{M}$, the run has odd length:

  $$C_0 \vdash C_1 \vdash C_2 \vdash C_3 \vdash \cdots \vdash C_n$$

  where n is odd.

After adding those states, the following holds for every word w:

- If $\mathcal{M}$ accepts w, then the run is finite and has odd length.

Some observations

Let $\mathcal{M}$ be a Turing machine.

- Add a "new" state q_{loop} such that instead of entering the q_{rej}, $\mathcal{M}$ enters q_{loop} and loops forever.
- Add some states, so that for every word w accepted by $\mathcal{M}$, the run has odd length:

  $$C_0 \vdash C_1 \vdash C_2 \vdash C_3 \vdash \cdots \vdash C_n$$

  where n is odd.

After adding those states, the following holds for every word w:

- If $\mathcal{M}$ accepts w, then the run is finite and has odd length.
- If $\mathcal{M}$ does not w, then the run is infinite.

Some observations – continued

Recall that the states of a Turing machines $\mathcal{M}$ are represented as numbers written in binary form. Thus, the run (1) can be viewed as a string over the alphabet $\{\vdash, 0, 1, \tilde{\sqcup}, [,]\}$, where we write $[i]$ to represent the state in the configuration.

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor\mathcal{M}\rfloor\$w$, construct $\mathcal{G}_1$ and $\mathcal{G}_2$ such that:

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, construct $\mathcal{G}_1$ and $\mathcal{G}_2$ such that:

- If $\mathcal{M}\$w \in$ HALT, then $L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ contains exactly one word:

$$C_0 \ \vdash \ C_1^r \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n^r$$

  where C_i^r denotes the reverse of C_i and

$$C_0 \ \vdash \ C_1 \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n$$

  is the run of $\mathcal{M}$ on w.

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, construct $\mathcal{G}_1$ and $\mathcal{G}_2$ such that:

- If $\mathcal{M}\$w \in$ HALT, then $L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ contains exactly one word:

  $$C_0 \ \vdash \ C_1^r \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n^r$$

  where C_i^r denotes the reverse of C_i and

  $$C_0 \ \vdash \ C_1 \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n$$

  is the run of $\mathcal{M}$ on w.
- If $\mathcal{M}\$w \notin$ HALT, then $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$.

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, construct $\mathcal{G}_1$ and $\mathcal{G}_2$ such that:

- If $\mathcal{M}\$w \in$ HALT, then $L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ contains exactly one word:

  $$C_0 \ \vdash \ C_1^r \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n^r$$

  where C_i^r denotes the reverse of C_i and

  $$C_0 \ \vdash \ C_1 \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n$$

  is the run of $\mathcal{M}$ on w.

- If $\mathcal{M}\$w \notin$ HALT, then $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$.

(Def.) We call the string: $C_0 \ \vdash \ C_1^r \ \vdash \ C_2 \ \vdash \ C_3^r \ \vdash \ \cdots \ \vdash \ C_n^r$

the reverse representation of the run: $C_0 \ \vdash \ C_1 \ \vdash \ C_2 \ \vdash \ C_3 \ \vdash \ \cdots \ \vdash \ C_n$.

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \vdash \ \cdots \ \vdash \ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \vdash \cdots \vdash u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \vdash \cdots \vdash u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \vdash \ \cdots \ \vdash \ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

(c) $u_{i-1} \vdash u_i^r$, for each odd i in between 1 and n.

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \vdash \ \cdots \ \vdash \ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

(c) $u_{i-1} \vdash u_i^r$, for each odd i in between 1 and n.

(d) $u_{i-1}^r \vdash u_i$, for each even i in between 1 and n.

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \vdash \ \cdots \ \vdash \ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

(c) $u_{i-1} \vdash u_i^r$, for each odd i in between 1 and n.

(d) $u_{i-1}^r \vdash u_i$, for each even i in between 1 and n.

(e) The last string u_n contains $[q_{\mathrm{acc}}]$.

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \vdash \ \cdots \ \vdash \ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

(c) $u_{i-1} \vdash u_i^r$, for each odd i in between 1 and n.

(d) $u_{i-1}^r \vdash u_i$, for each even i in between 1 and n.

(e) The last string u_n contains $[q_{\mathrm{acc}}]$.

There is an algorithm where on input $\lfloor \mathcal{M} \rfloor$, it constructs a CFG $\mathcal{G}_1$ such that $\mathcal{G}_1$ generates the strings that satisfies conditions (a), (b) and (c).

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \vdash \ \cdots \ \vdash \ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

(c) $u_{i-1} \vdash u_i^r$, for each odd i in between 1 and n.

(d) $u_{i-1}^r \vdash u_i$, for each even i in between 1 and n.

(e) The last string u_n contains $[q_{\mathrm{acc}}]$.

There is an algorithm where on input $\lfloor \mathcal{M} \rfloor$, it constructs a CFG $\mathcal{G}_1$ such that $\mathcal{G}_1$ generates the strings that satisfies conditions (a), (b) and (c).

There is an algorithm where on input $\lfloor \mathcal{M} \rfloor$, it constructs a CFG $\mathcal{G}_2$ such that $\mathcal{G}_2$ generates the strings that satisfies conditions (d) and (e).

The construction of $\mathcal{G}_1$ and $\mathcal{G}_2$

A string $u_0 \ \vdash\ u_1 \ \vdash\ u_2 \ \vdash\ u_3 \ \vdash\ \cdots \ \vdash\ u_n$ is the reverse representation of the run of $\mathcal{M}$ on w, if:

(a) n is an odd number, i.e., the symbol $\vdash$ appears even number of times.

(b) u_0 is the initial configuration of $\mathcal{M}$ on w.

(c) $u_{i-1} \vdash u_i^r$, for each odd i in between 1 and n.

(d) $u_{i-1}^r \vdash u_i$, for each even i in between 1 and n.

(e) The last string u_n contains $[q_{\mathrm{acc}}]$.

There is an algorithm where on input $\lfloor \mathcal{M} \rfloor$, it constructs a CFG $\mathcal{G}_1$ such that $\mathcal{G}_1$ generates the strings that satisfies conditions (a), (b) and (c).

There is an algorithm where on input $\lfloor \mathcal{M} \rfloor$, it constructs a CFG $\mathcal{G}_2$ such that $\mathcal{G}_2$ generates the strings that satisfies conditions (d) and (e).

(See Note 8 for the details.)

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Add some new states to $\mathcal{M}$ so that:
  $\mathcal{M}$ accepts w iff the run of $\mathcal{M}$ on w is finite and has odd length.
- Construct $\mathcal{G}_1$ that generates words satisfying conditions (a), (b) and (c).
- Construct $\mathcal{G}_2$ that generates words satisfying conditions (d) and (e).
- Output $\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor$.

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Add some new states to $\mathcal{M}$ so that:
  $\mathcal{M}$ accepts w iff the run of $\mathcal{M}$ on w is finite and has odd length.
- Construct $\mathcal{G}_1$ that generates words satisfying conditions (a), (b) and (c).
- Construct $\mathcal{G}_2$ that generates words satisfying conditions (d) and (e).
- Output $\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor$.

$L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ contains the reverse representation of the accepting run of $\mathcal{M}$ on w.

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Add some new states to $\mathcal{M}$ so that:
  $\mathcal{M}$ accepts w iff the run of $\mathcal{M}$ on w is finite and has odd length.
- Construct $\mathcal{G}_1$ that generates words satisfying conditions (a), (b) and (c).
- Construct $\mathcal{G}_2$ that generates words satisfying conditions (d) and (e).
- Output $\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor$.

$L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ contains the reverse representation of the accepting run of $\mathcal{M}$ on w.

Thus,

$$\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT} \qquad \text{if and only if} \qquad L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$$

The reduction HALT $\leqslant_m$ CFL-Intersection

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Add some new states to $\mathcal{M}$ so that:
  $\mathcal{M}$ accepts w iff the run of $\mathcal{M}$ on w is finite and has odd length.
- Construct $\mathcal{G}_1$ that generates words satisfying conditions (a), (b) and (c).
- Construct $\mathcal{G}_2$ that generates words satisfying conditions (d) and (e).
- Output $\lfloor \mathcal{G}_1 \rfloor \$ \lfloor \mathcal{G}_2 \rfloor$.

$L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ contains the reverse representation of the accepting run of $\mathcal{M}$ on w.

Thus,

$$\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT} \qquad \text{if and only if} \qquad L(\mathcal{G}_1) \cap L(\mathcal{G}_2) \neq \emptyset$$

Hence, CFL-Intersection is undecidable.

CFL universality

CFL-Universality

Input: A CFG $\mathcal{G} = \langle \Sigma, V, R, S \rangle$ where $\Sigma = \{0, 1\}$.

Task: Output `True`, if $L(\mathcal{G}) = \Sigma^*$. Otherwise, output `False`.

CFL universality

CFL-Universality

Input: A CFG $\mathcal{G} = \langle \Sigma, V, R, S \rangle$ where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}) = \Sigma^*$. Otherwise, output `False`.

Similar to CFL-Intersection, the problem CFL-Universality can be viewed as language.

CFL universality

CFL-Universality

Input:	A CFG $\mathcal{G} = \langle \Sigma, V, R, S \rangle$ where $\Sigma = \{0, 1\}$.
Task:	Output `True`, if $L(\mathcal{G}) = \Sigma^*$. Otherwise, output `False`.

Similar to CFL-Intersection, the problem CFL-Universality can be viewed as language.

Theorem 8.9
The problem CFL-Universality *is undecidable.*

Proof that CFL-Universality is undecidable

The proof is similar to Theorem 8.8.

Proof that CFL-Universality **is undecidable**

The proof is similar to Theorem 8.8.

We describe an algorithm that does the following.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct a CFG $\mathcal{G}$ such that:
  $\mathcal{G}$ generates all strings that are *not*(!) the run of $\mathcal{M}$ on w.

Proof that CFL-Universality is undecidable

The proof is similar to Theorem 8.8.

We describe an algorithm that does the following.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct a CFG $\mathcal{G}$ such that:
  $\mathcal{G}$ generates all strings that are *not*(!) the run of $\mathcal{M}$ on w.

If $\lfloor \mathcal{M} \rfloor \$ w \notin$ HALT, then $L(\mathcal{G}) = \Sigma^*$.

Proof that CFL-Universality is undecidable

The proof is similar to Theorem 8.8.

We describe an algorithm that does the following.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct a CFG $\mathcal{G}$ such that:
  $\mathcal{G}$ generates all strings that are *not*(!) the run of $\mathcal{M}$ on w.

If $\lfloor \mathcal{M} \rfloor \$ w \notin \text{HALT}$, then $L(\mathcal{G}) = \Sigma^*$.

If $\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT}$, then $L(\mathcal{G}) \neq \Sigma^*$.

Proof that CFL-Universality is undecidable

The proof is similar to Theorem 8.8.

We describe an algorithm that does the following.

On input $\lfloor \mathcal{M} \rfloor \$ w$:

- Construct a CFG $\mathcal{G}$ such that:
  $\mathcal{G}$ generates all strings that are *not*(!) the run of $\mathcal{M}$ on w.

If $\lfloor \mathcal{M} \rfloor \$ w \notin \mathsf{HALT}$, then $L(\mathcal{G}) = \Sigma^*$.

If $\lfloor \mathcal{M} \rfloor \$ w \in \mathsf{HALT}$, then $L(\mathcal{G}) \neq \Sigma^*$.

Thus,

$$\lfloor \mathcal{M} \rfloor \$ w \in \mathsf{HALT} \qquad \text{if and only if} \qquad L(\mathcal{G}) \neq \Sigma^*$$

The construction of the CFG $\mathcal{G}$

A word $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \cdots \vdash u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

The construction of the CFG $\mathcal{G}$

A word $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \cdots \vdash u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

(C1) The symbol $\vdash$ appears even number of times.

The construction of the CFG $\mathcal{G}$

A word $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \cdots \vdash u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

(C1) The symbol $\vdash$ appears even number of times.

(C2) u_0 is not the initial configuration.

The construction of the CFG $\mathcal{G}$

A word $u_0 \ \vdash \ u_1 \ \vdash \ u_2 \ \vdash \ u_3 \ \cdots \ \vdash \ u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

(C1) The symbol $\vdash$ appears even number of times.

(C2) u_0 is not the initial configuration.

(C3) For some $0 \leqslant i \leqslant n$, the string u_i is not a configuration.
It does not contain a state or the states appear at least twice or the brackets [and] do not appear "properly" or inside the bracket [and] is not a state of $\mathcal{M}$.

The construction of the CFG $\mathcal{G}$

A word $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \cdots \vdash u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

(C1) The symbol $\vdash$ appears even number of times.

(C2) u_0 is not the initial configuration.

(C3) For some $0 \leqslant i \leqslant n$, the string u_i is not a configuration.
It does not contain a state or the states appear at least twice or the brackets [and] do not appear "properly" or inside the bracket [and] is not a state of $\mathcal{M}$.

(C4) For some $0 \leqslant i \leqslant n-1$, the string $u_i \vdash u_i$ is not according to the transitions of $\mathcal{M}$.

The construction of the CFG $\mathcal{G}$

A word $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \cdots \vdash u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

(C1) The symbol $\vdash$ appears even number of times.

(C2) u_0 is not the initial configuration.

(C3) For some $0 \leqslant i \leqslant n$, the string u_i is not a configuration.
It does not contain a state or the states appear at least twice or the brackets [and] do not appear "properly" or inside the bracket [and] is not a state of $\mathcal{M}$.

(C4) For some $0 \leqslant i \leqslant n-1$, the string $u_i \vdash u_i$ is not according to the transitions of $\mathcal{M}$.

(C5) For some $o \leqslant i \leqslant n-1$, the string u_i is not the reverse of u_{i+1} (disregarding the state symbol and the symbols next to the state in both u_i and u_{i+1}).

The construction of the CFG $\mathcal{G}$

A word $u_0 \vdash u_1 \vdash u_2 \vdash u_3 \cdots \vdash u_n$ is not the reverse representation of the run $\mathcal{M}$ on w, if at least one of the following holds.

(C1) The symbol $\vdash$ appears even number of times.

(C2) u_0 is not the initial configuration.

(C3) For some $0 \leqslant i \leqslant n$, the string u_i is not a configuration.
It does not contain a state or the states appear at least twice or the brackets [and] do not appear "properly" or inside the bracket [and] is not a state of $\mathcal{M}$.

(C4) For some $0 \leqslant i \leqslant n-1$, the string $u_i \vdash u_i$ is not according to the transitions of $\mathcal{M}$.

(C5) For some $o \leqslant i \leqslant n-1$, the string u_i is not the reverse of u_{i+1} (disregarding the state symbol and the symbols next to the state in both u_i and u_{i+1}).

(C6) The last string u_n does not contain q_{acc}.

The construction of the CFG $\mathcal{G}$ – continued

We can construct one CFG $\mathcal{G}_i$ that generates all the strings that satisfy one condition (Ci), where $1 \leqslant i \leqslant 6$.

The construction of the CFG $\mathcal{G}$ – continued

We can construct one CFG $\mathcal{G}_i$ that generates all the strings that satisfy one condition (Ci), where $1 \leqslant i \leqslant 6$.

It is useful to recall that CFL are closed union.

The construction of the CFG $\mathcal{G}$ – continued

We can construct one CFG $\mathcal{G}_i$ that generates all the strings that satisfy one condition (Ci), where $1 \leqslant i \leqslant 6$.

It is useful to recall that CFL are closed union.

The final CFG $\mathcal{G}$ generates $L(\mathcal{G}_1) \cup \cdots \cup L(\mathcal{G}_6)$.

The reduction HALT $\leqslant_T$ CFL-Universality

The following algorithm assumes that there is an algorithm for checking whether $L(\mathcal{G}) = \Sigma^*$.

The reduction HALT $\leqslant_T$ CFL-Universality

The following algorithm assumes that there is an algorithm for checking whether $L(\mathcal{G}) = \Sigma^*$.

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

The reduction HALT $\leqslant_T$ CFL-Universality

The following algorithm assumes that there is an algorithm for checking whether $L(\mathcal{G}) = \Sigma^*$.

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Construct the CFG $\mathcal{G}$ that generates words where at least one of (C1)–(C6) holds.

The reduction HALT $\leqslant_T$ CFL-Universality

The following algorithm assumes that there is an algorithm for checking whether $L(\mathcal{G}) = \Sigma^*$.

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Construct the CFG $\mathcal{G}$ that generates words where at least one of (C1)–(C6) holds.
- If $L(\mathcal{G}) = \Sigma^*$, then REJECT.
  If $L(\mathcal{G}) \neq \Sigma^*$, then ACCEPT.

The reduction HALT $\leqslant_T$ CFL-Universality

The following algorithm assumes that there is an algorithm for checking whether $L(\mathcal{G}) = \Sigma^*$.

On input $\lfloor \mathcal{M} \rfloor \$ w$, do the following.

- Construct the CFG $\mathcal{G}$ that generates words where at least one of (C1)–(C6) holds.
- If $L(\mathcal{G}) = \Sigma^*$, then REJECT.
  If $L(\mathcal{G}) \neq \Sigma^*$, then ACCEPT.

The algorithm is correct due to:

$$\lfloor \mathcal{M} \rfloor \$ w \in \text{HALT} \qquad \text{if and only if} \qquad L(\mathcal{G}) \neq \Sigma^*$$

To conclude:

CFL-Intersection and CFL-Universality are both undecidable.

To conclude:

CFL-Intersection and CFL-Universality are both undecidable.

Consider the following problem.

CFL-Subset

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.

Task: Output `True`, if $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$. Otherwise, output `False`.

To conclude:

CFL-Intersection and CFL-Universality are both undecidable.

Consider the following problem.

CFL-Subset

Input: Two CFG $\mathcal{G}_1 = \langle \Sigma, V_1, R_1, S_1 \rangle$ and $\mathcal{G}_2 = \langle \Sigma, V_2, R_2, S_2 \rangle$, where $\Sigma = \{0, 1\}$.
Task: Output `True`, if $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$. Otherwise, output `False`.

The following is a direct consequence of the undecidability of CFL-Universality.

Corollary 8.10
The problem CFL-Subset *is undecidable.*

End of Lesson 8