Lesson 7. Universal Turing machines and the halting problem

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 and assumptions

(Recall) A Turing machine is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}, \delta \rangle$.

Recall and assumptions

(Recall) A Turing machine is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}, \delta \rangle$.

From now on, we assume that $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$.

Recall and assumptions

(Recall) A Turing machine is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}, \delta \rangle$.

From now on, we assume that $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$.

We also assume that $Q = \{0, 1, \ldots, n\}$ for some positive integer n.

Recall and assumptions

(Recall) A Turing machine is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}, \delta \rangle$.

From now on, we assume that $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$.

We also assume that $Q = \{0, 1, \ldots, n\}$ for some positive integer n.

(Goal) To show that Turing machines can be represented as strings and there is an algorithm/TM that verifies whether a string represents a Turing machine.

The encoding of a Turing machine $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathbf{acc}}, q_{\mathbf{rej}}, \delta \rangle$

Each state $i \in Q$ can be written as a string in its binary form.

The encoding of a Turing machine $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathbf{acc}}, q_{\mathbf{rej}}, \delta \rangle$

Each state $i \in Q$ can be written as a string in its binary form.

Each transition $(i, a) \ \rightarrow \ (j, b, \alpha)$ in δ can be written as a string over the alphabet $\{\texttt{0},\ \texttt{1},\ (,\ ),\ \diamond, \rightarrow,\ \tilde{\sqcup},\ \texttt{L},\ \texttt{R}\}$.

The encoding of a Turing machine $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathbf{acc}}, q_{\mathbf{rej}}, \delta \rangle$

Each state $i \in Q$ can be written as a string in its binary form.

Each transition $(i, a) \rightarrow (j, b, \alpha)$ in δ can be written as a string over the alphabet $\{\mathtt{0},\ \mathtt{1},\ \mathtt{(},\ \mathtt{)},\ \diamond, \rightarrow,\ \tilde{\sqcup},\ \mathtt{L},\ \mathtt{R}\}$.

The intention is to use $\diamond$ to represent the comma, $\tilde{\sqcup}$ to represent $\sqcup$, and $\mathtt{L}$, $\mathtt{R}$ to represent `Left`, `Right`, respectively.

The encoding of a Turing machine $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathbf{acc}}, q_{\mathbf{rej}}, \delta \rangle$

Each state $i \in Q$ can be written as a string in its binary form.

Each transition $(i, a) \rightarrow (j, b, \alpha)$ in δ can be written as a string over the alphabet $\{0,\ 1,\ (,\ ),\ \diamond, \rightarrow,\ \tilde{\sqcup},\ \mathtt{L},\ \mathtt{R}\}$.

The intention is to use $\diamond$ to represent the comma, $\tilde{\sqcup}$ to represent $\sqcup$, and $\mathtt{L}$, $\mathtt{R}$ to represent $\mathtt{Left}$, $\mathtt{Right}$, respectively.

For example, a transition

$$(5, 0) \ \rightarrow \ (8, 1, \mathtt{Right})$$

is written as the string:

$$(101 \diamond 0) \ \rightarrow \ (1000 \diamond 1 \diamond \mathtt{R})$$

The generalization to multiple tape Turing machines

For 3-tape Turing machine, e.g., a transition

$$(7, 0, \sqcup, 1) \;\rightarrow\; (9, 1, 0, 1, \texttt{Right}, \texttt{Left}, \texttt{Left})$$

is written as the string:

$$(111 \diamond 0 \diamond \tilde{\sqcup} \diamond 1) \;\rightarrow\; (1001 \diamond 1 \diamond 0 \diamond 1 \diamond \texttt{R} \diamond \texttt{L} \diamond \texttt{L})$$

The encoding of a Turing machine, continued

The TM $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}, \delta \rangle$ can be written as a string:

$$\lfloor \Sigma \rfloor \;\#\; \lfloor \Gamma \rfloor \;\#\; \lfloor Q \rfloor \;\#\; \lfloor q_0 \rfloor \;\#\; \lfloor q_{\mathrm{acc}} \rfloor \;\#\; \lfloor q_{\mathrm{rej}} \rfloor \;\#\; \lfloor \delta \rfloor$$

where $\lfloor \cdot \rfloor$ denotes the string representing the component $\cdot$ and $\#$ the symbol separating two consecutive components.

The encoding of a Turing machine, continued

The TM $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\rm acc}, q_{\rm rej}, \delta \rangle$ can be written as a string:

$$\lfloor \Sigma \rfloor \;\#\; \lfloor \Gamma \rfloor \;\#\; \lfloor Q \rfloor \;\#\; \lfloor q_0 \rfloor \;\#\; \lfloor q_{\rm acc} \rfloor \;\#\; \lfloor q_{\rm rej} \rfloor \;\#\; \lfloor \delta \rfloor$$

where $\lfloor \cdot \rfloor$ denotes the string representing the component $\cdot$ and $\#$ the symbol separating two consecutive components.

For example, if $Q = \{0, \ldots, 45\}$, 0 is the initial state, 3 is $q_{\rm acc}$ and 4 is $q_{\rm rej}$, then the TM is written as a string:

$$\underbrace{0 \diamond 1}_{\lfloor \Sigma \rfloor} \;\#\; \underbrace{0 \diamond 1 \diamond \tilde{\sqcup}}_{\lfloor \Gamma \rfloor} \;\#\; \underbrace{45}_{\lfloor Q \rfloor} \;\#\; \underbrace{0}_{\lfloor q_0 \rfloor} \;\#\; \underbrace{3}_{\lfloor q_{\rm acc} \rfloor} \;\#\; \underbrace{4}_{\lfloor q_{\rm rej} \rfloor} \;\#\; \underbrace{\cdots\cdots\cdots\cdots\cdots\cdots}_{\text{the list of the transitions}}$$

The encoding of a Turing machine, continued

The TM $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}, \delta \rangle$ can be written as a string:

$$\lfloor \Sigma \rfloor \ \# \ \lfloor \Gamma \rfloor \ \# \ \lfloor Q \rfloor \ \# \ \lfloor q_0 \rfloor \ \# \ \lfloor q_{\mathrm{acc}} \rfloor \ \# \ \lfloor q_{\mathrm{rej}} \rfloor \ \# \ \lfloor \delta \rfloor$$

where $\lfloor \cdot \rfloor$ denotes the string representing the component $\cdot$ and $\#$ the symbol separating two consecutive components.

For example, if $Q = \{0, \ldots, 45\}$, 0 is the initial state, 3 is q_{acc} and 4 is q_{rej}, then the TM is written as a string:

$$\underbrace{0 \diamond 1}_{\lfloor \Sigma \rfloor} \ \# \ \underbrace{0 \diamond 1 \diamond \tilde{\sqcup}}_{\lfloor \Gamma \rfloor} \ \# \ \underbrace{45}_{\lfloor Q \rfloor} \ \# \ \underbrace{0}_{\lfloor q_0 \rfloor} \ \# \ \underbrace{3}_{\lfloor q_{\mathrm{acc}} \rfloor} \ \# \ \underbrace{4}_{\lfloor q_{\mathrm{rej}} \rfloor} \ \# \ \underbrace{\cdots\cdots\cdots\cdots\cdots}_{\text{the list of the transitions}}$$

(Note) Every TM (whose tape alphabet is $\Gamma = \{0, 1, \sqcup\}$) can be described as a string over the alphabet $\{0,\ 1,\ (,\ ),\ \diamond, \rightarrow,\ \tilde{\sqcup},\ \mathtt{L},\ \mathtt{R},\ \#\}$.

The 0-1 string representation of a Turing machine

Each of the symbols 0, 1, (,), $\diamond$, $\rightarrow$, $\tilde{\sqcup}$, L, R, $\#$ can be encoded as 0-1 string of length 4. For example,

symbol	the encoding
0	0000
1	0001
(	0010
)	0011
$\diamond$	0100

symbol	the encoding
$\rightarrow$	0101
$\tilde{\sqcup}$	0110
L	0111
R	1000
$\#$	1001

The 0-1 string representation of a Turing machine

Each of the symbols $\mathtt{0}$, $\mathtt{1}$, $($, $)$, $\diamond$, $\rightarrow$, $\tilde{\sqcup}$, $\mathtt{L}$, $\mathtt{R}$, $\#$ can be encoded as 0-1 string of length 4. For example,

symbol	the encoding
$\mathtt{0}$	0000
$\mathtt{1}$	0001
$($	0010
$)$	0011
$\diamond$	0100

symbol	the encoding
$\rightarrow$	0101
$\tilde{\sqcup}$	0110
$\mathtt{L}$	0111
$\mathtt{R}$	1000
$\#$	1001

(Def.) $\lfloor \mathcal{M} \rfloor$ denotes the 0-1 string obtained by such encoding.

We call $\lfloor \mathcal{M} \rfloor$ *the binary string representation* of the Turing machine $\mathcal{M}$, or *the description of $\mathcal{M}$*.

Verifying the description of a Turing machine

A string w represents a Turing machine, if it is of the form:

$$u_1 \# u_2 \# u_3 \# u_4 \# u_5 \# u_6 \# u_7$$

each string u_i satisfies the following.

- u_1 is $0 \diamond 1$ and u_2 is $0 \diamond 1 \diamond \tilde{\square}$.
- u_3 is an integer n (written in binary form) and u_4, u_5, u_6 are all some numbers (in binary form) between 0 and n.
- u_7 is a string that lists all the transitions: For every (i, a), there is *exactly one* (j, b, α) where

  $$(i \diamond a) \rightarrow (j \diamond b \diamond \alpha)$$

  appears in u_7.

(Note) We can write an algorithm/computer program that on input w, checks whether it satisfies all the properties above.

Verifying the description of a Turing machine – continued

Recalling the following encoding.

symbol	the encoding
0	0000
1	0001
(	0010
)	0011
$\diamond$	0100

symbol	the encoding
$\rightarrow$	0101
$\tilde{\sqcup}$	0110
L	0111
R	1000
#	1001

Verifying the description of a Turing machine – continued

Recalling the following encoding.

symbol	the encoding
0	0000
1	0001
(	0010
)	0011
$\diamond$	0100

symbol	the encoding
$\rightarrow$	0101
$\tilde{\sqcup}$	0110
L	0111
R	1000
#	1001

We can modify the program for verifying all the properties above when each of the symbols 0, 1, (,), $\diamond$, $\rightarrow$, $\tilde{\sqcup}$, L, R, # is encoded as 0-1 string above.

Verifying the description of a Turing machine – continued

Verifying the description of a Turing machine

Input: A string w over the alphabet $\{0,1\}$.

Task: Output `True`, if w is the description of a TM $\mathcal{M}$, i.e. $w = \lfloor \mathcal{M} \rfloor$ (under the 0-1 encoding shown in the table above)

Output `False`, otherwise.

Verifying the description of a Turing machine – continued

Verifying the description of a Turing machine

Input: A string w over the alphabet $\{0, 1\}$.

Task: Output `True`, if w is the description of a TM $\mathcal{M}$, i.e. $w = \lfloor \mathcal{M} \rfloor$
(under the 0-1 encoding shown in the table above)
Output `False`, otherwise.

Proposition 7.2

There is an algorithm $\mathcal{A}$ for the problem **Verifying the description of a Turing machine**.

Table of contents

Universal Turing machines

(Def.) A *universal Turing machine* (UTM) is a Turing machine $\mathcal{U}$ that on input $\lfloor \mathcal{M} \rfloor \$ w$, where $w \in \{0,1\}^*$, does the following.

- If $\mathcal{M}$ accepts w, then $\mathcal{U}$ accepts $\lfloor \mathcal{M} \rfloor \$ w$.
- If $\mathcal{M}$ rejects w, then $\mathcal{U}$ rejects $\lfloor \mathcal{M} \rfloor \$ w$.
- If $\mathcal{M}$ does not halt on w, then $\mathcal{U}$ does not halt on $\lfloor \mathcal{M} \rfloor \$ w$.

How a UTM $\mathcal{U}$ works

On input word u:

How a UTM $\mathcal{U}$ works

On input word u:

- Check if u is of the form:

$$v\$w$$

  where $v, w \in \{0,1\}^*$.

How a UTM $\mathcal{U}$ works

On input word u:

- Check if u is of the form:

$$v\$w$$

where $v, w \in \{0,1\}^*$.

- Check if v is indeed the description of a TM $\mathcal{M}$, i.e.,

$$v = \lfloor \mathcal{M} \rfloor$$

If it is not, REJECT. Otherwise, continue.

How a UTM $\mathcal{U}$ works

On input word u:

- Check if u is of the form:

$$v\$w$$

  where $v, w \in \{0,1\}^*$.

- Check if v is indeed the description of a TM $\mathcal{M}$, i.e.,

$$v \ = \ \lfloor \mathcal{M} \rfloor$$

  If it is not, REJECT. Otherwise, continue.

- Construct the initial configuration C of $\mathcal{M}$ on w.

How a UTM $\mathcal{U}$ works

On input word u:

- Check if u is of the form:

  $$v\$w$$

  where $v, w \in \{0,1\}^*$.

- Check if v is indeed the description of a TM $\mathcal{M}$, i.e.,

  $$v \ = \ \lfloor \mathcal{M} \rfloor$$

  If it is not, REJECT. Otherwise, continue.

- Construct the initial configuration C of $\mathcal{M}$ on w.
- while (C is not a halting configuration):

How a UTM $\mathcal{U}$ works

On input word u:

- Check if u is of the form:

  $$v\$w$$

  where $v, w \in \{0,1\}^*$.

- Check if v is indeed the description of a TM $\mathcal{M}$, i.e.,

  $$v = \lfloor \mathcal{M} \rfloor$$

  If it is not, REJECT. Otherwise, continue.

- Construct the initial configuration C of $\mathcal{M}$ on w.
- while (C is not a halting configuration):
  - Compute the next configuration of C (by accessing the transition of $\mathcal{M}$).

How a UTM $\mathcal{U}$ works

On input word u:

- Check if u is of the form:

$$v\$w$$

  where $v, w \in \{0,1\}^*$.

- Check if v is indeed the description of a TM $\mathcal{M}$, i.e.,

$$v \;=\; \lfloor \mathcal{M} \rfloor$$

  If it is not, REJECT. Otherwise, continue.

- Construct the initial configuration C of $\mathcal{M}$ on w.
- while (C is not a halting configuration):
  - Compute the next configuration of C (by accessing the transition of $\mathcal{M}$).
- If C is an accepting configuration, ACCEPT.
  If C is a rejecting configuration, REJECT.

How a UTM $\mathcal{U}$ works – continued

The UTM $\mathcal{U}$ basically constructs the run of $\mathcal{M}$ on w.

How a UTM $\mathcal{U}$ works – continued

The UTM $\mathcal{U}$ basically constructs the run of $\mathcal{M}$ on w.

It is similar to the proof of Theorem 6.1.

How a UTM $\mathcal{U}$ works – continued

The UTM $\mathcal{U}$ basically constructs the run of $\mathcal{M}$ on w.

It is similar to the proof of Theorem 6.1.

(Note) A UTM is defined according to the 0-1 encoding of the symbols 0, 1, (,), $\diamond$, $\rightarrow$, $\tilde{\square}$, L, R, $\#$.

How a UTM $\mathcal{U}$ works – continued

The UTM $\mathcal{U}$ basically constructs the run of $\mathcal{M}$ on w.

It is similar to the proof of Theorem 6.1.

(Note) A UTM is defined according to the 0-1 encoding of the symbols 0, 1, $($, $)$, $\diamond$, $\rightarrow$, $\tilde{\square}$, $\mathtt{L}$, $\mathtt{R}$, $\#$.

Different encoding yields different UTM.

An analogy of a UTM

As an analogy, a compiler is a sort of UTM.

An analogy of a UTM

As an analogy, a compiler is a sort of UTM.

For example, C++ compiler accepts as input a C++ program P and the input w for P. Then, it “simulates” the program P on input w.

An analogy of a UTM

As an analogy, a compiler is a sort of UTM.

For example, C++ compiler accepts as input a C++ program P and the input w for P. Then, it "simulates" the program P on input w.

We can likewise view Python compiler as a UTM.

An analogy of a UTM

As an analogy, a compiler is a sort of UTM.

For example, C++ compiler accepts as input a C++ program P and the input w for P. Then, it “simulates” the program P on input w.

We can likewise view Python compiler as a UTM.

We can view C++ syntax and Python syntax are two different descriptions/encodings of Turing machines.

An analogy of a UTM

As an analogy, a compiler is a sort of UTM.

For example, C++ compiler accepts as input a C++ program P and the input w for P. Then, it "simulates" the program P on input w.

We can likewise view Python compiler as a UTM.

We can view C++ syntax and Python syntax are two different descriptions/encodings of Turing machines.

C++ compiler can only run C++ programs (i.e., programs written in C++ syntax) and Pyhton compiler can only run Python programs.

An analogy of a UTM

As an analogy, a compiler is a sort of UTM.

For example, C++ compiler accepts as input a C++ program P and the input w for P. Then, it "simulates" the program P on input w.

We can likewise view Python compiler as a UTM.

We can view C++ syntax and Python syntax are two different descriptions/encodings of Turing machines.

C++ compiler can only run C++ programs (i.e., programs written in C++ syntax) and Pyhton compiler can only run Python programs.

A PC/laptop/phone is also UTM in the sense that it takes as input a program/app P and an input w, and it simulates P on w. (though it makes the impression that you run P yourself.)

Table of contents

The halting problem

In the following we assume that the description of Turing machines is defined under a fixed encoding.

The halting problem

In the following we assume that the description of Turing machines is defined under a fixed encoding.

Consider the following languages

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

The halting problem

In the following we assume that the description of Turing machines is defined under a fixed encoding.

Consider the following languages

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

We can view HALT'_0 is the "complement" of HALT_0.

The halting problem

In the following we assume that the description of Turing machines is defined under a fixed encoding.

Consider the following languages

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

We can view HALT'_0 is the "complement" of HALT_0.

Technically this is not "correct", since the complement of HALT_0 includes strings that are not the description of Turing machines.

However, recall that we have an algorithm that checks whether a string is really the description of a Turing machine (Proposition 7.2), which we can use to accept/reject strings that are not descriptions of Turing machines.

The languages HALT and HALT_0

$$\begin{aligned} \mathsf{HALT} &:= \{\lfloor \mathcal{M} \rfloor \$ w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}. \\ \mathsf{HALT}_0 &:= \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ accepts } \lfloor \mathcal{M} \rfloor\}. \end{aligned}$$

The languages HALT and HALT_0

$$\begin{aligned}\mathsf{HALT} &:= \{\lfloor\mathcal{M}\rfloor\$w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}.\\ \mathsf{HALT}_0 &:= \{\lfloor\mathcal{M}\rfloor \mid \mathcal{M} \text{ accepts } \lfloor\mathcal{M}\rfloor\}.\end{aligned}$$

Proposition 7.5

The language $HALT_0$ and HALT are recognizable.

The languages HALT and HALT_0

$$\begin{aligned} \mathsf{HALT} &:= \{\lfloor \mathcal{M} \rfloor \$ w \mid \mathcal{M} \text{ accepts } w \text{ where } w \in \{0,1\}^*\}. \\ \mathsf{HALT}_0 &:= \{\lfloor \mathcal{M} \rfloor \mid \mathcal{M} \text{ accepts } \lfloor \mathcal{M} \rfloor\}. \end{aligned}$$

Proposition 7.5

The language HALT_0 and HALT are recognizable.

(Proof) Use the UTM $\mathcal{U}$.

The language $\mathbf{HALT}_0'$

Theorem 7.6

$HALT_0'$ is undecidable.

The language HALT'_0

Theorem 7.6
$HALT'_0$ is undecidable.

(Proof) Suppose to the contrary that HALT'_0 is decidable.

Let $\mathcal{B}$ be the TM that decides HALT'_0.

The language $\mathbf{HALT}_0'$

Theorem 7.6
$HALT_0'$ is undecidable.

(Proof) Suppose to the contrary that HALT_0' is decidable.

Let $\mathcal{B}$ be the TM that decides HALT_0'.

- If $\mathcal{B}$ accepts $\lfloor \mathcal{B} \rfloor$.

The language HALT_0'

Theorem 7.6
$HALT_0'$ is undecidable.

(Proof) Suppose to the contrary that HALT_0' is decidable.

Let $\mathcal{B}$ be the TM that decides HALT_0'.

- If $\mathcal{B}$ accepts $\lfloor \mathcal{B} \rfloor$.

  Since $\mathcal{B}$ decides HALT_0', this means $\lfloor \mathcal{B} \rfloor \in \mathsf{HALT}_0'$.
  By the definition of HALT_0', $\mathcal{B}$ does not accept $\lfloor \mathcal{B} \rfloor$. A contradiction.

The language HALT_0'

Theorem 7.6
$HALT_0'$ is undecidable.

(Proof) Suppose to the contrary that HALT_0' is decidable.

Let $\mathcal{B}$ be the TM that decides HALT_0'.

- If $\mathcal{B}$ accepts $\lfloor \mathcal{B} \rfloor$.

  Since $\mathcal{B}$ decides HALT_0', this means $\lfloor \mathcal{B} \rfloor \in \mathsf{HALT}_0'$.
  By the definition of HALT_0', $\mathcal{B}$ does not accept $\lfloor \mathcal{B} \rfloor$. A contradiction.

- If $\mathcal{B}$ rejects $\lfloor \mathcal{B} \rfloor$.

The language HALT_0'

Theorem 7.6
$HALT_0'$ is undecidable.

(Proof) Suppose to the contrary that HALT_0' is decidable.

Let $\mathcal{B}$ be the TM that decides HALT_0'.

- If $\mathcal{B}$ accepts $\lfloor \mathcal{B} \rfloor$.

  Since $\mathcal{B}$ decides HALT_0', this means $\lfloor \mathcal{B} \rfloor \in \mathsf{HALT}_0'$.
  By the definition of HALT_0', $\mathcal{B}$ does not accept $\lfloor \mathcal{B} \rfloor$. A contradiction.

- If $\mathcal{B}$ rejects $\lfloor \mathcal{B} \rfloor$.

  Since $\mathcal{B}$ decides HALT_0', this means $\lfloor \mathcal{B} \rfloor \notin \mathsf{HALT}_0'$.
  By the definition of HALT_0', $\mathcal{B}$ accepts $\lfloor \mathcal{B} \rfloor$. A contradiction.

The language HALT_0'

Theorem 7.6
$HALT_0'$ is undecidable.

(Proof) Suppose to the contrary that HALT_0' is decidable.

Let $\mathcal{B}$ be the TM that decides HALT_0'.

- If $\mathcal{B}$ accepts $\lfloor\mathcal{B}\rfloor$.

  Since $\mathcal{B}$ decides HALT_0', this means $\lfloor\mathcal{B}\rfloor \in \mathsf{HALT}_0'$.
  By the definition of HALT_0', $\mathcal{B}$ does not accept $\lfloor\mathcal{B}\rfloor$. A contradiction.

- If $\mathcal{B}$ rejects $\lfloor\mathcal{B}\rfloor$.

  Since $\mathcal{B}$ decides HALT_0', this means $\lfloor\mathcal{B}\rfloor \notin \mathsf{HALT}_0'$.
  By the definition of HALT_0', $\mathcal{B}$ accepts $\lfloor\mathcal{B}\rfloor$. A contradiction.

Both cases yield contradiction. Thus, HALT_0' is undecidable.

The language $\mathbf{HALT}_0'$ – continued

Theorem 7.6 actually states the same thing as Theorem 0.1 in Lesson 0.

The language HALT'_0 – continued

Theorem 7.6 actually states the same thing as Theorem 0.1 in Lesson 0.

The only difference is that Theorem 7.6 is formulated in term of the Turing machines while Theorem 0.1 is formulated in term of the C++ programs.

Some easy corollaries

Note that if HALT_0 and HALT are decidable, then HALT_0' is also decidable.
Thus,

Some easy corollaries

Note that if HALT_0 and HALT are decidable, then HALT_0' is also decidable. Thus,

Corollary 7.7

$HALT_0$ and $HALT$ are undecidable.

Some easy corollaries

Note that if HALT_0 and HALT are decidable, then HALT_0' is also decidable. Thus,

Corollary 7.7

$HALT_0$ and HALT are undecidable.

Moreover, HALT_0' is the complement of HALT_0 and HALT_0 is recognizable. Thus,

Corollary 7.8

The language $HALT_0'$ is not recognizable.

To conclude:

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

To conclude:

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

We have proved:

- HALT_0 and HALT are undecidable, but recognizable.

To conclude:

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

We have proved:

- HALT_0 and HALT are undecidable, but recognizable.
- HALT_0' is not recognizable.

End of Lesson 7