Lesson 7: Universal Turing machines and the halting problem

Theme: Universal Turing machines and the halting problem.

1 The string representation of a Turing machine

Recall that a Turing machine is a system \( M = \langle \Sigma, \Gamma, Q, q_0, \text{acc}, \text{rej}, \delta \rangle \). In the following we will assume that \( \Sigma = \{0,1\} \) and \( \Gamma = \{0,1,\sqcup\} \). Without loss of generality, we may also assume that \( Q = \{0,1,\ldots,n\} \) for some positive integer \( n \).

We note the following.

- Each state \( i \in Q \) can be written as a string in its binary form.
- Each transition \( (i,a) \to (j,b,\alpha) \in \delta \) can be written as a string over the alphabet:

\[
\{0, 1, (,), \circ, \rightarrow, \sqcup, \text{L, R}\}
\]

where the symbol \( \circ \) represents the comma, \( \sqcup \) represents \( \sqcup \), and \( \text{L, R} \) represent Left, Right, respectively. For example, a transition \( (5,\sqcup) \to (8,1,\text{Right}) \) is written as the string:

\[
(101 \circ \sqcup) \to (1000 \circ 1 \circ \text{R})
\]

So, the whole system \( M = \langle \Sigma, \Gamma, Q, 0, \text{acc}, \text{rej}, \delta \rangle \) can be written as a string:

\[
|\Sigma| \# |\Gamma| \# |Q| \# |0| \# |\text{acc}| \# |\text{rej}| \# |\delta|
\]

where \( |\cdot| \) denotes the string representing the component \( \cdot \) and \( \# \) the symbol separating two consecutive components.

For example, if \( M \) is a 1-tape TM where \( Q = \{0, \ldots, 45\} \), 0 is the initial state, 3 is \( \text{acc} \) and 4 is \( \text{rej} \), it is written as a string:

\[
0 \circ 1 \# 0 \circ 1 \circ \sqcup \# 45 \# 0 \# 3 \# 4 \# \text{the list of the transitions}
\]

Recall that we use the symbol \( \circ \) to represent a comma here. Note also that we do not list all the states \( 0, \ldots, 45 \). It suffices to write 45 to indicate that the states are all the numbers between 0 and 45.

This shows that every 1-tape TM (whose tape alphabet is \( \Gamma = \{0,1,\sqcup\} \)) can be described as a string over the alphabet \( \{0, 1, (,), \circ, \rightarrow, \sqcup, \text{L, R, #}\} \). Each of these symbols can be further encoded as 0-1 string of length 4. For example, 0 is encoded as 0000, 1 as 0001 and so on, as shown in the following table.

<table>
<thead>
<tr>
<th>symbol</th>
<th>the encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>(</td>
<td>0010</td>
</tr>
<tr>
<td>)</td>
<td>0011</td>
</tr>
<tr>
<td>\circ</td>
<td>0100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>symbol</th>
<th>the encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>\rightarrow</td>
<td>0101</td>
</tr>
<tr>
<td>\sqcup</td>
<td>0110</td>
</tr>
<tr>
<td>\text{L}</td>
<td>0111</td>
</tr>
<tr>
<td>\text{R}</td>
<td>1000</td>
</tr>
<tr>
<td>#</td>
<td>1001</td>
</tr>
</tbody>
</table>

We denote by \( |M| \) the binary string obtained by such encoding and we call \( |M| \) the binary string representation of the Turing machine \( M \), or the description of \( M \).
Remark 7.1 Note that we can easily extend the definition of $\lfloor M \rfloor$ for TM $M$ with multiple tapes. In this course, $\lfloor M \rfloor$ denotes the string representation of $M$ where $M$ can be any TM with multiple tapes.

Note that a string $w$ over the alphabet $\{0, 1, (, ), \circ, \rightarrow, \sqcup, L, R, \#\}$ is the string representation of a Turing machine, if it is of the form:

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

that is, the symbol $\#$ appears exactly 6 times and each string $u_i$ satisfies the following.

- $u_1$ is $0 \circ 1$.
- $u_2$ is $0 \circ 1 \circ \sqcup$.
- $u_3$ is an integer $n$ (written in binary form).
- $u_4$, $u_5$, $u_6$ are all the binary form of some numbers between 0 and $n$.
- $u_7$ is a string such that for every $(i, a) \in \{0, \ldots, n\} \times \{0, 1, \sqcup\}$, there is exactly one $(j, b, \alpha) \in \{0, \ldots, n\} \times \{0, 1, \sqcup\} \times \{L, R\}$ such that $(i \circ a) \rightarrow (j \circ b \circ \alpha)$ appears in $u_7$.

We can easily write an algorithm/computer program that on input string $w$ over the alphabet $\{0, 1, (, ), \circ, \rightarrow, \sqcup, L, R, \#\}$, it checks whether $w$ satisfies all these properties. In similar manner, we can also write an algorithm/computer program that on input string $w$ over $\{0, 1\}$, it checks whether $w$ is the binary string representation of a Turing machine. This observation is summarized formally as the following proposition.

**Proposition 7.2** There is an algorithm $A$ for the following problem.

<table>
<thead>
<tr>
<th>Verifying the description of a Turing machine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A string $w$ over the alphabet ${0, 1}$.</td>
</tr>
<tr>
<td><strong>Task:</strong> Output $\textbf{True}$, if $w$ is indeed the description of a TM $M$, i.e. $w = \lfloor M \rfloor$.</td>
</tr>
<tr>
<td>Output $\textbf{False}$, otherwise.</td>
</tr>
</tbody>
</table>

Note that the algorithm $A$ depends on the 0-1 encoding of the symbols 0, 1, (, ), $\circ$, $\rightarrow$, $\sqcup$, L, R and $. Under different encoding, a different algorithm is needed for verifying the description of a Turing machine. Throughout this course we will assume the fixed encoding shown in the table above. Hence, we also assume a fixed algorithm $A$ for verifying the description of a Turing machine.

2 Universal Turing machines

**Definition 7.3** A universal Turing machine (UTM) is a Turing machine $U$ that on input $\lfloor M \rfloor$#$w$, where $w \in \{0, 1\}^*$, does the following.

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

Intuitively, a UTM $U$ works as follows. On input $\lfloor M \rfloor$#$w$, it simulates $M$ on $w$, i.e., it constructs the run of $M$ on $w$. The way it works is actually similar to the proof of Theorem 6.1 except that now the TM $M$ is given as part of the input. More precisely, on input word $u$, it does the following.
• 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 \( M \), i.e., \( v = [M] \), by using the algorithm in Proposition 7.2.
  If it is not, REJECT. Otherwise, continue.
• Construct the initial configuration of \( M \) on \( w \) and store it as a string \( C \).
• while (\( C \) is not a halting configuration):
  – Compute the next configuration of \( C \) (by accessing the transition of \( M \)).
• If \( C \) is an accepting configuration, ACCEPT. If \( C \) is a rejecting configuration, REJECT.

It is obvious that: (i) if \( M \) accepts \( w \), then \( U \) accepts \([M]\$w\); (ii) if \( M \) rejects \( w \), then \( U \) rejects \([M]\$w\) and (iii) if \( M \) does not halt on \( w \), then \( U \) does not halt on \([M]\$w\).

Similar to algorithm \( A \) in Proposition 7.2, a UTM is defined according to the encoding of the descriptions of the Turing machines. Since we assume that we only use one fixed encoding, throughout this course we will also assume a fixed UTM \( U \).

Remark 7.4 At this point we would like to clarify on the meaning of the phrases “run a TM \( M \) on \( w \)” and “simulate a TM \( M \) on \( w \).”

Intuitively, “run a TM \( M \) on \( w \)” means that we view a TM \( M \) as a procedure/function (written, say, in C++) and we “call” it with input \( w \). On the other hand, “simulate a TM \( M \) on \( w \)” essentially means that we construct the run of \( M \) on \( w \) (assuming that we have access to the transitions of \( M \)).

3 The halting problem

We define the following languages:

\[
\text{HALT} := \{[M] \$w \mid M \text{ accepts } w \text{ where } w \in \{0, 1\}^*\}
\]

\[
\text{HALT}_0 := \{[M] \mid M \text{ accepts } [M]\}
\]

\[
\text{HALT}'_0 := \{[M] \mid M \text{ does not accept } [M]\}
\]

Note that we can use the UTM \( U \) to recognize the language \( \text{HALT} \) and we can also easily modify the UTM \( U \) to recognize the language \( \text{HALT}_0 \). We state this formally as the following proposition.

Proposition 7.5 The language \( \text{HALT}_0 \) and \( \text{HALT} \) are recognizable.

Theorem 7.6 \( \text{HALT}_0 \) is undecidable.

Proof. Suppose to the contrary that \( \text{HALT}'_0 \) is decidable. Let \( B \) be the TM that decides \( \text{HALT}'_0 \). We examine whether \( B \) accept its own description \([B]\). There are two cases.

• If \( B \) accepts \([B]\).
  Since \( B \) decides \( \text{HALT}'_0 \), this means \([B] \in \text{HALT}'_0 \). By the definition of \( \text{HALT}'_0 \), \( B \) does not accept \([B]\). A contradiction.

*Sometimes these two phrases are used interchangeably, since the end results (in terms of accept/reject) are usually the same. However, the two processes have different run time.
• If $\mathcal{B}$ rejects $\lfloor \mathcal{B} \rfloor$.

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

Both cases yield contradiction. Thus, there is no such TM $\mathcal{B}$ that decides $\text{HALT}_0'$, i.e., $\text{HALT}_0'$ is undecidable.

We should note that 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.

**Corollary 7.7** $\text{HALT}_0$ and $\text{HALT}$ are undecidable.

**Proof.** It follows immediately from Theorem 7.6

Recall that if both $L$ and its complement $\overline{L} = \Sigma^* - L$ are recognizable, then both are decidable. Then, the following corollary follows immediately from Proposition 7.5 and Theorem 7.6.

**Corollary 7.8** The language $\text{HALT}_0'$ is not recognizable.