LECTURE Notes #25: The Halting Problem;
Turing-Acceptable vs. Turing-Decidable

THE HALTING PROBLEM  (Chapter 5, Section 5.3, p. 251)

• What is the Halting Problem?

To determine for an arbitrary given Turing Machine M and input w,

whether M will eventually halt on input w

• Why do we care? (i.e what’s the motivation)

To find an example of a Turing-undecidable Language.

Because by Church’s Thesis, (algorithm \(\Leftrightarrow\) TM), such a language has a membership problem that cannot be solved (computed) by any kind of algorithm whatsoever.

• We already know that such languages exist:

By a counting argument:

• Every computable (recursive) language is decided by a TM

• There are only countably many TMs

• There are uncountably many languages

• Most languages are not Turing-decidable / computable / recursive
Two simple basic facts about Turing-decidable and Turing-acceptable Languages

1. If L is Turing-decidable then L is Turing-acceptable

Proof:
If M decides L, then the TM:
accepts/semidecides L

i.e if M decides L, we enter into either state y or n.

- If M halts in y, this machine halts
(because if w \( \in \) L, M goes to state y and halts)

- If M halts in n, this machine Runs Forever moving right
(goes into infinite loop if halts in n because w \( \notin \) L,
that is, the TM above doesn’t halt \( \iff \) doesn’t accept)

If L is Turing-decidable then L is Turing-acceptable

(i.e If L is decided by some TM, M, then L is also accepted by some TM,
namely the one above).

2. If L is Turing-decidable then so is \( \overline{L} \)

Proof:
This machine goes to a no (n) state when M ended up in a yes (y) state,
and vice-a-versa.

Therefore there is also a TM which decides \( \overline{L} \).
• Is every Turing-acceptable set Turing-decidable?

• This *would* be the case if there were an algorithm to solve the
  **Halting problem**
  Given any arbitrary TM, M, and any arbitrary input w, does M halt on input w?
  (i.e. if for any TM, M and input, w, we always knew that, or when M would halt,
  then we could modify M to make it decide \( L = \{ \text{any } w \} \))

• In particular, if \( L = L(M) \) were a turing-acceptable set, we could *decide* \( L \)
  by (a TM version of) the following algorithm:
  Given w, **solve the halting problem** for M and w,
  and go to state y or n accordingly (i.e. y if Turing-acceptable, else n).

• So the real question is:
  **Is the Halting problem Solvable?**

  In trying to answer this question fully, let’s consider the issues of
  *Turing-acceptable* and *Turing-decidable* in more detail.
**Turing Acceptable :**

1. M accepts \( w \in \Sigma^* \) if M *halts* on input \( w \)

2. Language L is Turing-acceptable if there is *some* TM, M that accepts it
   - That is, given a countably infinite number of TMs \( M_1, M_2, \ldots, M_i, \ldots \) and a language L,
     
     L is Turing-acceptable if at least one of the \( M_i \) accepts (halts on)
     
     any \( w \in L : L = \{ w \in \Sigma^* : M_i \text{ accepts } w \} \)

But the question is:

How do we know or discover which \( M_i \) (if any) accepts L ?

If we start testing \( M_1, M_2, \ldots \) if \( M_1 \) doesn’t accept L it will never halt. Yet, how do we know that if we run \( M_1 \) long enough it won’t eventually halt on \( w \) ? So the issue comes down to : we would like to be able to accurately *predict* whether a Machine will eventually halt on input \( w \). For if it were possible to predict for *any TM* and *any input string* whether or not that TM would eventually halt on that input then every Turing-acceptable language would also be Turing-decidable.

*e.g. :*

1. Given machine M, input w, design machine \( M^* \) such that
2. \( M^* \) performs calculations to *predict* the eventual outcome of M’s computations on input w (i.e solve the Halting Problem)
3. If M would halt on input w
   
   then go to the \( y \) state
   
   else go to the \( n \) state

So the question of whether every T.A language is T.D comes down to whether there is a TM that can predict the outcome of computations by arbitrary TMs, M, on arbitrary inputs, w.
**Turing-acceptable Vs Turing-decidable:**

- Consider the language $K_0 = \{ \rho(M) \rho(w) : \text{TM M accepts input w} \}$
  
  1. $K_0$ is Turing-acceptable (by definition): 
     
     There is a machine, $M_0$ (a variant of U) 
     which accepts input $= \rho(M) \rho(w)$ such that M accepts input w. 
  
  2. So if M accepts $w \in L$ then $M_0$ also accepts $\rho(M) \rho(w) \in K_0$ 
  
  3. $K_0$ is like an infinite dictionary: 
     
     $K_0 = \{ \rho(M_1) \rho(w_{11}), \rho(M_1) \rho(w_{12}),..., \rho(M_1) \rho(w_{1n}),... \rho(M_2) \rho(w_{21}), \rho(M_2) \rho(w_{22}),... \}$ 
     
     … a dictionary which contains the answer to the question: 
     
     Does arbitrary TM M accept arbitrary input w? 
     
     Because to answer the question we 
     
     1. look in the dictionary to see if M is paired (appears) with input w 
     2. If M does appear with w then M accepts w, else M doesn’t accept w 
  
  4. So $K_0$ is a *formalized version* of the Halting Problem 
     
     - because for any arbitrary M and w, we could just examine $K_0$ to see if $\rho(M) \rho(w) \in K_0$ 
     - but since $K_0$ is infinite, if we don’t find $\rho(M) \rho(w)$ we will never know if we have searched long enough (just as we would never know for M, w, if we had let M compute long enough that it might have eventually accepted w) 
  
  5. However, if we could answer the question in general: 
     
     is $\rho(M) \rho(w) \in K_0$ 
     
     then we could solve the Halting Problem $\iff$ we could decide $K_0$ 
  
  6. But $K_0$ is not Turing-decidable $\Rightarrow$ the Halting Problem is not solvable 
  
  7. For *specific* M, w, we could predict whether M will halt on input w 
     
     e.g.: 
     
     1. Could see if M has any halt states 
     2. For real simple M, w 
     
     But there is no *completely general* method that correctly decides all cases.
Proof of The Halting Problem:

- Let the language $K_o = \{ \rho(M)\rho(w) : \text{TM } M \text{ accepts input } w \} \ (\rho \Rightarrow \text{encoding.})$

- $K_o$ is the formalized version of the halting problem
  (i.e expresses any input $w$ for any TM, $M$, both as inputs to $U$)

- Also, by the way $K_o$ is defined, $K_o$ is Turing-acceptable because the TM, $M_o$, accepts $K_o$. (i.e. $M_o$, like the universal machine, $U$, takes strings of the form $\rho(M)\rho(w)$, and simulates whatever $M$ would do - which (in the case of $K_o$ is halt/accept.)

- Hence,
  Every Turing-acceptable language is Turing-decidable iff the particular Turing-acceptable language, $K_o$ is Turing-decidable.
  (i.e $K_o$ is the hardest (most general) Turing-acceptable language - so if we show $K_o$ is Turing-decidable, that can cover all cases in a general way)

- However, $K_o$ is not Turing-decidable

Proof:

1. If the language $K_o$ is Turing-decidable $\Rightarrow$ the language $K_1$ is Turing-decidable where $K_1 = \{ \rho(M) : \text{TM, } M \text{ accepts input } \rho(M) \}$
  (i.e. $K_1 = \text{set of encoded representations, } \rho(M) \text{ of } M : M \text{ accepts } \rho(M)$)

- In other words, if the language $K_o$ is Turing-decidable, $K_1$ is Turing-decidable also, because we could construct a TM to transform input $w = \rho(M) \in K_1$ to $\#w \rho(w)\# = \#\rho(M)\rho(\rho(M))\#$ and then pass control to $M_o$.

- So to answer the question “Is $\rho(M) \in K_1$?”, (i.e. to decide $K_1$), just ask the question “Is $\rho(M)\rho(\rho(M)) \in K_o$?” (i.e. decide $K_0$).
  (That is, decidable just means being able to answer the question,”Is $w \in L$?”)

∴ To show that $K_o$ is not Turing-decidable, it is sufficient to show that $K_1$ is not Turing-decidable
2. If \( K_1 \) is Turing-decidable \( \implies \) \( K_1 (K_1 = K_1 \text{ complement}) \) is Turing-decidable
   (shown previously)

3. But \( K_1 \) is not even Turing-acceptable:
   
   **Proof:**
   
   \( K_1 = \{ w : w = \rho(M) \text{ for some TM, M that does not accept } \rho(M) \} \)
   
   *that is, w was not the encoding \( \rho(M) \) of any TM, M*
   
   Suppose, there were a TM, \( M^* \), which accepts
   
   \( K_1 = \{ \rho(M) : M \text{ does not accept } \rho(M) \} \)
   
   Then is \( \rho(M^*) \in K_1 \)?

   | If \( \rho(M^*) \in K_1 \) \( \iff \rho(M^*) \notin K_1 \iff M^* \text{ does not accept } \rho(M^*) \) —— (i) |
   | --- | --- |
   | But since \( M^* \) accepts \( K_1 \) (by def) |
   | \( and \) \( \rho(M^*) \) is a member of \( K_1 \) |
   | then \( M^* \) must accept \( \rho(M^*) \) because |
   | \( \rho(M^*) \) is a member of the set \( K_1 \) of \( \rho(M) \)s which \( M^* \) accepts |
   | So, \( \rho(M^*) \in K_1 \iff M^* \text{ accepts } \rho(M^*) \), |
   | which is in contradiction to (i) above. |
   | \( \therefore \) there exists no such \( M^* \). |

4. Finally, \( K_1 \) not T.A. \( \implies \) \( K_1 \) not T.D. \( \implies \) \( K_1 \) not T.D. \( \implies \) \( K_0 \) not T.D.
The previous proof actually makes use of the diagonalization principle:

\[ K_1 = \{ \rho(M) : \text{TM, M accepts input } \rho(M) \} \]

Look at \( K_1 \) as a table where \( y \Rightarrow \text{set member}, \ n \Rightarrow \text{not a set member} \):

(For example, for \( i = j \), \( K_1 = \{ \rho(M_0), \rho(M_1), \rho(M_3) : M_i \text{ accepts } \rho(M_j) \} \) )

<table>
<thead>
<tr>
<th></th>
<th>( \rho(M_0) )</th>
<th>( \rho(M_1) )</th>
<th>( \rho(M_2) )</th>
<th>( \rho(M_3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_0 )</td>
<td>( y )</td>
<td>( n )</td>
<td>( n )</td>
<td>( y )</td>
</tr>
<tr>
<td>( M_1 )</td>
<td>( y )</td>
<td>( y )</td>
<td>( n )</td>
<td>( n )</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>( n )</td>
<td>( n )</td>
<td>( n )</td>
<td>( n )</td>
</tr>
<tr>
<td>( M_3 )</td>
<td>( y )</td>
<td>( y )</td>
<td>( y )</td>
<td>( y )</td>
</tr>
</tbody>
</table>

- Table entry corresponding to \( (M_i, \rho(M_j)) \) is \( y \) iff \( M_i \) accepts \( \rho(M_j) \)

- \( K_1 \) is represented by the complement of the diagonal which must be different from every row, and since \( K_1 \) is different from every row, there can’t be an \( M_i \) to accept it.

For example \( K_1 = \{ \rho(M_2) : M_2 \text{ does not accept } \rho(M_2) \} \), and none of the machines above accept only \( \{ \rho(M_2) : M_2 \text{ does not accept } \rho(M_2) \} \). That is, if \( M_2 \) accepts \( K_1 \) the way \( K_1 \) is defined.

\[ \therefore K_1 \text{ is not Turing-acceptable since } K_1 = \text{exactly the set of } \rho(M) : \text{no TM accepts } \rho(M). \]

- Two theorems resulting from the previous proof:
  1. Not every Turing-acceptable language is Turing-decidable
  2. The complements of some Turing-acceptable languages are not Turing-acceptable