Proving Undecidability through Self-Reference and Diagonalization Methods

Lecturer: Debanjan Bhowmik, Indian Institute of Technology Delhi
Scribed by: Het Patel (2019EE0484)
and Debanjan Bhowmik

Abstract

A language is said to be decidable if there exists a Turing Machine (TM) such that any string belonging to that language is accepted by the TM (the final state of the TM for that computation is an 'accept' state) and any string not belonging to that language (the final state for that computation is a 'reject' state) is rejected by the TM. If no such TM which follows both these criteria exists for a language, then the language is undecidable. Here, we prove by contradiction (we first assume the language to be decidable/ such a TM exists and then show a contradiction) that the languages HALT_{TM} and A_{TM} (as defined in the text below) are undecidable. We use two different methods for each proof (by contradiction): self-reference method and diagonal method (and argue they are essentially similar ideas). Proving that HALT_{TM} is undecidable is also same as showing that a general program/ algorithm to solve the halting problem for all possible program-input pairs cannot exist.

1 Proof that HALT_{TM} is undecidable

1.1 Problem Statement

Formal problem statement: Prove the following:

Theorem 1: The following language is undecidable: $\text{HALT}_{TM} = \{< M, w >$, M is a TM and M halts/ doesn't infinitely loop upon input w to the TM M $\}$

Alternative Problem Statement: Does a TM, say U, exist which takes as its input description of another another TM M and a string w and says if M halts on w or not (goes in an infinite loop)? Going by the Church Turing thesis, a computer program/ algorithm is a set of steps that can be executed on a corresponding TM. So this question is same as asking: can we have a general program/ algorithm to solve the halting problem for all possible program-input

pairs (the halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever)?

1.2 Proof: By Self-Reference Method

Let us assume HALT_{TM} is decidable. This means there exists a TM U (decider) that takes as its input $< M, w >$, where M is the description of the TM and w is a string, and provides the following output:

$$\begin{cases} \textbf{Accept} & \text{if } M \text{ halts upon input } w \text{ to } M \text{ (final state for } M\text{: accept/ reject)} \\ \textbf{Reject} & \text{if } M \text{ goes into an infinite loop/ doesn't halt upon input } w \end{cases}$$

If decider U exists, then there exists a TM M' such that when $< M >$ (description of a TM M) is applied as an input to M', M' calls the aforementioned decider U and passes on to it an input in the form: $< M, < M >>$. The output of M' is 'accept' if U's output is 'accept'. But M' goes in an infinite loop if U's output is 'reject'. Thus, for the input $< M >$ applied to M', M' responds as follows:

$$\begin{cases} \textbf{Accept} & \text{If } M \text{ accepts/ rejects } < M > \\ \textbf{Loop} & \text{If } M \text{ goes in an infinite loop/ doesn't halt for input } < M > \end{cases}$$

Note: M' is used here just for the sake of understanding, it's not needed for the proof. Later, in our diagonalization-method-based proof, we show that M' is equivalent to the diagonal in our table.

If decider U exists, also there exists another TM N which does the reverse of M', i.e, when $< M >$ is applied as an input to N, N calls the aforementioned decider U and passes on to it an input in the form: $< M, < M >>$. The output of N is 'accept' if U's output is 'reject'. But N goes in an infinite loop if U's output is 'accept'. Thus, for the input $< M >$ applied to N, N responds as follows:

$$\begin{cases} \textbf{Loop} & \text{If } M \text{ accepts/ rejects } < M > \\ \textbf{Accept} & \text{If } M \text{ goes in an infinite loop/ doesn't halt for input } < M > \end{cases}$$

Note: While U is a decider, M' and N are not deciders (they don't have to be), and so they can go in infinite loops.

What happens when the input to TM N is its own description $< N >$? Now N calls the aforementioned decider U and passes on to it an input: $< N, < N >>$. If N goes in an infinite loop/ doesn't halt for input $< N >$, N provides the output **Accept** (for the input $< N >$ applied to N). **This a contradiction.** If N accepts/ rejects $< N >$, N goes into an infinite **loop** (for the input $< N >$ applied to N). **This is also a contradiction.**

So, for either case, there is a contradiction, and so TM N can't exist. But if decider U exists, TM N must exist. So decider U can't exist. Hence, our initial assumption that HALT_{TM} is decidable is wrong. Thus, we prove by contradiction that HALT_{TM} is undecidable.

1.3 Proof: By Cantor's diagonalization method

We first show some simple proofs (lemmas) in set theory using Cantor's diagonalization method to demonstrate how all that lead to our final proof using the same diagonalization method that HALT_{TM} is undecidable.

Lemma 1: A set of all binary strings (each character/ digit of the string is 0 or 1) of infinite length is an uncountable set.

Proof: We prove this by contradiction as well. Let's assume the set is countable, i.e., each string is associated with a natural number/ an element of $\mathbb{N}$. Then we can write down the following table (neither number of rows or number of columns is finite for the table, '...' means there are various other elements here):

String number	Digit 1	Digit 2	Digit 3	...	...	Digit N	...
1	0	1	0	...	...	...	...
2	1	1	0	....	...	...	...
3	0	0	1	...	...	...	...
...	...	...	...	...	...	...	...
...							
...	...	...	...	...	...	...	...
...							
N	...	...	...	...	...	x	...
...	...	...	...	...	...	...	...

Table 1: A table of all binary strings of infinite length

In Table 1 above, String 1 is: 010..., String 2 here is 110..., String 3 is 001..., and so on. Now, if we take the diagonal from Table 1, it is also a binary string (011....) and can be placed on the table, say at row N. Say, the digit N is x, where x can be either 0 or 1.

Now, if we instead create another string from this diagonal where each digit of the string is the complement of that at the same place on the diagonal (so for Table 1, it is: 110...,100...,000..., and so on), then it cannot be placed in Table 1 because for any string N, the N-th digit is $\bar{x}$ as opposed to x. But if the set of binary strings of infinite length is countable, then we must be able to place even this string in the table because it belongs to the set. **Because of this contradiction, we conclude that the set of binary strings of infinite length is uncountable.**

Lemma 2: A set of all binary strings of infinite length, where some characters/ digits are fixed to 0 is also an uncountable set.

Proof: Even if some characters/ digits of each binary string are fixed to 0 (and even if infinite such characters are fixed to 0), the string is still infinite in length and the remaining character places/ digits (places in the string where the characters 1 or 0 occur) can still be mapped to $\mathbb{N}$. So a similar table as Table 1 can still be used to show that this new set is also uncountable.

Note: Proving uncountability with this modification to binary strings is important because in our actual proof, the Turing machines do not take as input all possible finite-length strings (from the given alphabet) but only certain finite-length strings which correspond to different TM descriptions. Just like all strings of finite length (countable) can be correlated to all characters places/ digit numbers (1, 2, 3,...) of a binary string in Table 1, strings of finite length corresponding to different TM descriptions can be correlated to the remaining character places of a binary string where some character places have 0s fixed in them. Thus we can correlate the table that we will use for our digonalization procedure in the actual proof with the table for proving this lemma, and in turn Table 1.

Lemma 3: A set of all strings, from a fixed alphabet, of finite length is countable.

Proof: For a fixed alphabet, a set of all strings of finite length comprising characters from that alphabet is a countable set (countably infinite). This is because for every string length (1, 2, 3,...) there is a finite number of strings. Number of possible strings of string length l = (Number of characters in the alphabet)^l. So the strings can be mapped to $\mathbb{N}$. Hence, this set is countable.

Note that following the logic we used in Lemma 2, even if we remove some strings (even an infinite number of them) from this set, the new set is still countably infinite.

Lemma 4: A set of all Turing machines is countable.

Proof: A Turing Machine (TM) is defined as the following 7-tuple: (states, alphabet for characters of the input string, tape alphabet, transition table, start state, an accept state, a reject state). Thus a TM can be expressed as a string of finite length with all the aforementioned information in the aforementioned order. Hence, a set of all TMs is essentially an infinite set of strings of finite length (not all strings, some have been removed, see Lemma 3 above). Hence, it is countably infinite (follow Lemma 3 above).

Theorem 1: The following language is undecidable: $\text{HALT}_{TM} = \{< M, w >$, M is a TM and M halts/ doesn't infinitely loop upon input w to the TM M $\}$

Proof: Now we come to our main proof. Since TMs form a countable set (Lemma 4), the following table can be constructed, with each row corresponding to a TM and each column corresponding to a TM description. Each element (i, j) of the table corresponds to what TM M_i does when description of TM M_j ($< M_j >$) is fed to the TM M_i as an input string (i= 1, 2, 3,...; j=1, 2, 3,...). There are three possibilities: accept, reject, goes in an infinite loop/ doesn't halt. The table is shown below:

Turing Machine	$< M_1 >$	$< M_2 >$	$< M_3 >$	...	...	$< M_N >$	...
M_1	accept	loop	reject	...	...	...	...
M_2	loop	loop	accept	....	...	...	...
M_3	accept	accept	reject	...	...	...	...
...	...	...	...	...	...	...	...
...							
...	...	...	...	...	...	...	...
...							
M_N	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Table 2: A table where each element (i, j) of the table corresponds to what TM M_i does when description of TM M_j ($< M_j >$) is fed to the TM M_i as an input string. (i= 1, 2, 3,...; j=1, 2, 3,...)

If HALT_{TM} is decidable, it means that there exists a decider U which takes as its input $< M_i, < M_j >>$ and yields the following output:

$$\begin{cases} \textbf{Accept} & \text{if } M_i \text{ halts upon input } < M_j > \text{ to } M_i \text{ (final state for } M_i\text{: accept/ reject)} \\ \textbf{Reject} & \text{if } M_i \text{ goes into an infinite loop/ doesn't halt upon input } < M_j > \end{cases}$$

Corresponding to Table 2, the decider U's table is then as follows (Table 3):

Turing Machine	$< M_1 >$	$< M_2 >$	$< M_3 >$	...	...	$< M_N >$	...
M_1	accept	reject	accept	...	...	...	...
M_2	reject	reject	accept	....	...	...	...
M_3	accept	accept	accept	...	...	...	...
...	...	...	...	...	...	...	...
...							
...	...	...	...	...	...	...	...
...							
M_N	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Table 3: A table for the decider U corresponding to HALT_{TM} (assuming HALT_{TM} is decidable

Using the diagonal in Table 3, we can construct TM M' (this is equivalent to M' in the self-reference method) which takes as an input $< M$i> and acts as

follows:

$$\begin{cases} \textbf{Accept} & \text{If } M_i \text{ accepts/ rejects } < M_i > \\ \textbf{Loop} & \text{If } M_i \text{ goes in an infinite loop/ doesn't halt for input } < M_i > \end{cases}$$

'Loop' here means M' goes in an infinite loop/ doesn't halt (again, M' doesn't need to be a decider).

Thus, from Table 2 and Table 3, M'($< M_1 >$)= accept, M'($< M_2 >$)= loop, M'($< M_3 >$)= accept,.... Hence, it is to be noted that M' is not exactly the diagonal of Table 3 but derived from it ('accept' in the diagonal is 'accept' for M' and 'reject' in the diagonal is 'loop' for M').

Just like in Lemma 1 we took the diagonal in Table 1 and took its complement (flipped 0s to 1s and 1s to 0s), here also we can do the same for Table 3 and construct another TM M_N (this is equivalent to our TM N in the self-reference method). M_N takes as an input $< M$i$>$ and does the opposite of M'. Thus M_N acts as follows:

$$\begin{cases} \textbf{Loop} & \text{If } M_i \text{ accepts/ rejects } < M_i > \\ \textbf{Accept} & \text{If } M_i \text{ goes in an infinite loop/ doesn't halt for input } < M_i > \end{cases}$$

Thus, from Table 2 and Table 3, N($< M_1 >$)= loop, N($< M_2 >$)= accept, N($< M_3 >$)= loop,.... Just like complement of the diagonal in Table 1 cannot be placed in Table 1 (Lemma 1 and 2), TM M_N cannot be placed in Table 3. If M_N is placed at the Nth row of Table 3 as shown, TM M_N accepts $< M_N >$ if M_N goes in an infinite loop upon input $< M_N >$. **This is a contradiction.** Similarly, TM M_N goes in an infinite loop upon input $< M_N >$ if M_N accepts/ rejects input $< M_N >$. **This is also a contradiction.** But since M_N is a TM, it should correspond to a row in Table 3. But since it can't due to the contradiction, we conclude that decider U can't exist.

Thus, we prove by Cantor's diagonalization technique that decider U doesn't exist and thus HALT_{TM} is undecidable. We note that this proof is essentially the same as the proof through the self-reference method, with the diagonal element of Table 3 for U corresponding to TM M' in the self-reference method (Section 1.2) and the complement of the diagonal element of Table 3 corresponding to TM N in the self-reference method (Section 1.2).

2 Proof that A_{TM} is undecidable

2.1 Problem Statement

Formal problem statement: Prove the following:

Theorem 2: The following language is undecidable: $\text{A}_{TM} = \{< M, w >$, M is a TM and M accepts input w to the TM M $\}$

Alternative Problem Statement: Does a TM exist which takes as its input description of another another TM M and a string w and accepts the pair

$< M, w >$ if TM M accepts w and rejects the pair $< M, w >$ if TM M rejects/ goes in an infinite loop for w?

2.2 Proof: By Self-Reference Method

Let us assume A_{TM} is decidable. This means there exists a TM V (decider) that takes as its input $< M, w >$, where M is the description of the TM and w is a string, and provides the following output:

$$\begin{cases} \textbf{Accept} & \text{if } M \text{ accepts } w \\ \textbf{Reject} & \text{if } M \text{ rejects/ goes in an infinite loop for } w \end{cases}$$

If decider V exists, also there exists another TM P: when $< M >$ is applied as an input to P, P calls the aforementioned decider V and passes on to it an input in the form: $< M, < M >>$. The output of P is 'accept' if V's output is 'reject'. The output of P is 'reject' if V's output is 'accept'. Thus, for the input $< M >$ applied to P, P responds as follows:

$$\begin{cases} \textbf{Reject} & \text{If } M \text{ accepts } < M > \\ \textbf{Accept} & \text{If } M \text{ goes in an infinite loop/ rejects } < M > \end{cases}$$

What happens when the input to TM P is its own description $< P >$? Now P calls the aforementioned V and passes on to it an input: $< P, < P >>$. If P goes in an infinite loop/ doesn't halt for input $< P >$, P provides the output **Accept** (for the input $< N >$ applied to N). **This a contradiction.** If P rejects $< P >$, then P provides the output **Accept**. **This also a contradiction.** If P accepts $< P >$, P provides the output **Reject**. **This also a contradiction.**

So, in all cases, there is a contradiction, and so TM P can't exist. But if decider V exists, TM P must exist. So decider V can't exist. Hence, our initial assumption that A_{TM} is decidable is wrong. Thus, we prove by contradiction that A_{TM} is undecidable.

2.3 Proof: By Cantor's diagonalization method

Since TMs form a countable set (Lemma 4), the following table can be constructed, with each row corresponding to a TM and each column corresponding to a TM description. Each element (i, j) of the table corresponds to what TM M_i does when description of TM M_j ($< M_j >$) is fed to the TM M_i as an input string (i= 1, 2, 3,...; j=1, 2, 3,...). There are three possibilities: accept, reject, goes in an infinite loop/ doesn't halt. The table is shown below:

If A_{TM} is decidable, it means that there exists a decider V which takes as its input $< M_i, < M_j >>$ and yields the following output:

$$\begin{cases} \textbf{Accept} & \text{if } M_i \text{ accepts input } < M_j > \\ \textbf{Reject} & \text{if } M_i \text{ rejects input } < M_j > \text{ or goes to an infinite loop for } < M_j > \end{cases}$$

Corresponding to Table 4, the decider V's table is then as follows (Table 5):

Just like in Lemma 1 we took the diagonal in Table 1 and took its complement (flipped 0s to 1s and 1s to 0s), here also we can do the same for Table 5 and

Turing Machine	$< M_1 >$	$< M_2 >$	$< M_3 >$	...	...	$< M_P >$	...
M_1	accept	loop	reject	...	...	...	...
M_2	loop	loop	accept	....	...	...	...
M_3	accept	accept	reject	...	...	...	...
...	...	...	...	...	...	...	...
...							
...	...	...	...	...	...	...	...
...							
M_P	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Table 4: A table where each element (i, j) of the table corresponds to what TM M_i does when description of TM M_j ($< M_j >$) is fed to the TM M_i as an input string. (i= 1, 2, 3,...; j=1, 2, 3,...)

Turing Machine	$< M_1 >$	$< M_2 >$	$< M_3 >$	...	...	$< M_P >$	...
M_1	accept	reject	reject	...	...	...	...
M_2	reject	reject	accept	....	...	...	...
M_3	accept	accept	reject	...	...	...	...
...	...	...	...	...	...	...	...
...							
...	...	...	...	...	...	...	...
...							
M_P	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

Table 5: A table for the decider V corresponding to A_{TM} (assuming $HALT_{TM}$ is decidable

construct another TM M_P (this is equivalent to our TM P in the self-reference method). M_P takes as an input $< M_i >$ and acts as follows:

$$\begin{cases} \textbf{Accept} & \text{if } M_i \text{ rejects input } < M_i > \text{ or goes to an infinite loop for } < M_i > \\ \textbf{Reject} & \text{if } M_i \text{ accepts input } < M_i > \end{cases}$$

Just like complement of the diagonal in Table 1 cannot be placed in Table 1 (Lemma 1 and 2), TM M_P cannot be placed in Table 5. If M_P is placed at the Pth row of Table 5 as shown, we end up in contradictions for all posible cases: TM M_P accepts $< M_P >$ if M_P rejects input $< M_P >$, TM M_P rejects $< M_P >$ if M_P accepts input $< M_P >$, TM M_P accepts $< M_P >$ if M_P goes in an infinite loop upon input $< M_P >$.

Thus, we prove by Cantor's diagonalization technique that decider V doesn't exist and thus A_{TM} is undecidable.

3 References

1. Michael Sipser, *Introduction to the Theory of Computation*, 3^{rd} Edition, Cengage Learning, June 2012

2. Dexter C. Kozen, *Automata and Computability*, Springer, 1997

3. Dan Gusfield, *'Theory of Computation' lectures*, Fall 2011, Department of Computer Science, School of Engineering, UC Davis