Lecture 4

A proof that an object exists without constructing it

In the previous lecture we saw a non-constructive proof that there are non-recognizable languages. In the following, we give a constructive proof that there are undecidable languages.

Sipser, 4.2

Theorem 4.11: The language $L_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$ is undecidable.

Proof: Suppose for the sake of contradiction that L_{TM} is decidable. Then, there must be a Turing machine H that is the decider of L_{TM}, i.e., if $x \in L_{TM}$, then $H(x)$ outputs "accept" and if $w \notin L_{TM}$ then $H(x)$ outputs "reject".

What does it mean when $H(x)$ outputs "accept"?

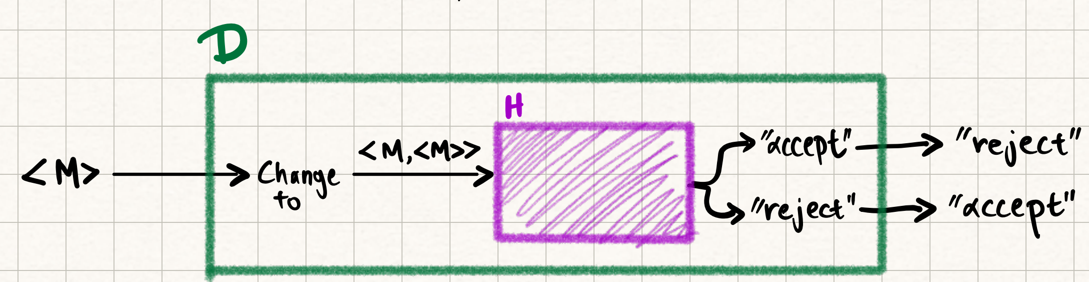
↳ It means that the string x which is the binary encoding of pair M, w, i.e., $x = \langle M, w \rangle$, is in the language L_{TM}. Therefore, if we run M with input w, M outputs "accept".

Analogously, when $H(x)$ outputs reject, then $x \notin L_{TM}$. Therefore, for $x = \langle M, w \rangle$ if we run M with input w, then M does not accept *

* M can either reject or loop. No guarantee which case we are at.

In summary, $H(\langle M, w \rangle) = \begin{cases} \text{"accept"}, & \text{if } M \text{ accepts } w \\ \text{"reject"}, & \text{if } M \text{ does not accept } w. \end{cases}$

We now use this (hypothetical) H, to construct a new TM D with input $\langle M \rangle$.



More formally, D = "On input $\langle M \rangle$, where M is a TM:
1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs."

* Notice that compilers are "machines"/programs that take as an input the description of another "machine"/program. Much like the input of H, that is $\langle M, \langle M \rangle \rangle$.

? What happens when we run D with input its own description $\langle D \rangle$?

We already know that

$$D(\langle M \rangle) = \begin{cases} \text{"accept"}, & \text{if } M \text{ does not accept input } \langle M \rangle. \\ \text{"reject"}, & \text{if } M \text{ outputs "accept" on input } \langle M \rangle. \end{cases}$$

Now swap $\langle M \rangle$ for $\langle D \rangle$ in the above expression.

$$D(\langle D \rangle) = \begin{cases} \text{"accept"}, & \text{if } D \text{ does not accept input } \langle D \rangle. \\ \text{"reject"}, & \text{if } D \text{ outputs "accept" on input } \langle D \rangle. \end{cases}$$

We constructed a paradox, therefore, neither TM D nor TM H can exist. Contradiction. ∎

Remark: The above proof can be seen under the lens of the diagonilization method.

- Suppose that we build a table that lists all the possible Turing machines as rows and all the possible input strings as columns

TMs \ Inputs	x_1	x_2	x_3	...
M_1	accept		accept	
M_2		accept		
M_3	accept	accept		
⋮				

As a next step, we
① remove all the input strings that do not encode a Turing machine
② We also reorder the columns to follow the row ordering
③ Populate the cells using the output of decider H. → Thus, even if $M_i(x_j)$ loops, decider $H(\langle M_i, x_j \rangle)$ will output "reject"

TMs \ Inputs	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...
M_1	accept	reject	accept	
M_2	reject	accept	reject	
M_3	accept	accept	reject	
⋮				

- Since the rows depict all possible TMs, one of them must be D.

TMs \ Inputs	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...
M_1	accept	reject	accept	
M_2	reject	accept	reject	
M_3	accept	accept	reject	
⋮				
D				
⋮				

Next, let's analyze how D behaves on each column.
On input $\langle M_1 \rangle$, outputs the opposite of $M_1(\langle M_1 \rangle)$
⋮
On input $\langle M_i \rangle$, outputs the opposite of $M_i(\langle M_i \rangle)$

But these are the outputs of the diagonal!

So,

TMs \ Inputs	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$	...
M_1	accept	reject	accept			
M_2	reject	accept	reject			
M_3	accept	accept	reject			
⋮	≠	≠	≠	⋱		
D	reject	reject	accept		?	
⋮						

This entry needs to be the opposite of itself. Contradiction.

Recall that last lecture's argument about non-recognizable languages was non-constructive. In the following, we will see a constructive proof for non-recognizability.

A natural first guess is to work with L_{TM} (from Theorem 4.11), which we know is undecidable. It is easy to see that L_{TM} is Turing-recognizable. Simply create a new machine M' that internally runs/simulates the input machine M on the input string w. If M accepts, then M' also accepts. If M rejects, M' also rejects, we are indifferent to the case where M' loops because M' is supposed to only recognize the language (as opposed to decide).

Definition: A complement of a language is the language consisting of all strings that are not in the language.

*Complement of L is denoted as $\overline{L}$.

Definition: A language is co-Turing-recognizable if it is the complement of a Turing recognizable language.

Theorem 4.22: A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable

Proof: Since it is an if and only if, we have to prove both directions.
① L is decidable $\Rightarrow$ L is Turing-recognizable and co-Turing-recognizable

② L is Turing-recognizable and co-Turing recognizable $\Rightarrow$ L is decidable

For ①: If L is decidable, then there exists a decider M. This machine M accepts all strings that are in the language, therefore, it can act as a recognizer which proves that L is Turing-recognizable. If we create a new machine M′ that simulates M internally and flips its output, then M′ accepts all the strings that the decider M rejects which that L is co-Turing-recognizable.

For ②: Since L is Turing-recognizable, there exists an M_1 that recognizes $\underline{L}$.
Since L is co-Turing-recognizable, there exists an M_2 that recognizes $\overline{L}$.
We propose a new TM M^* that uses M_1 and M_2 and decides L.

M^* = "On input w:
1. Alternate between a transition in $M_1(w)$ and a transition in $M_2(w)$.
2. If M_1 accepts first, then M^* accepts. If M_2 accepts first, then M^* rejects."

Finally, we have to show that M^* decides L. Every string is either in L (so M_1 accepts it) or in $\bar{L}$ (so M_2 accepts it). Notice that since M_1 and M_2 are recognizers, one of them halts and accepts no matter what the input is. Since M^* halts whenever M_1 or M_2 accepts, M^* always halts. Finally, it accepts all $w \in L$ and rejects all $w \notin L$. Thus, M^* is a decider so L is decidable. ■

Corollary 4.23: $\bar{L}_{TM}$ is not Turing-recognizable.

* Recall that L_{TM} is Turing-recognizable

Proof: We showed that L_{TM} is Turing-recognizable. If its complement $\bar{L}_{TM}$ were Turing-recognizable too, then L_{TM} would be decidable (from Theorem 4.22). But since we already proved in Theorem 4.11 that L_{TM} is not decidable, it must be that $\bar{L}_{TM}$ is not Turing-recognizable. ■

Reducability (Sipser, Chapter 5.1, Section 5.1 up to "Reductions via Computation Histories")

We showed the existence of a computationally unsolvable proben (i.e., deciding L_{TM}) on our most powerful computational model, the Turing machine. In the following, we see a methodology, called reducability, that capitalizes on previously proven unsolvable problems, to prove that new problems are also unsolvable.

A **reduction** is a way of converting a problem A to another problem B so that a solution to B can be used to solve A.

Attention: Reducability can be used for solving a problem but it can also be used to prove that a problem is not solvable.

- Suppose that we know how to solve B (so, colored green), but we don't know how to solve A (so, colored gray). If we find a reduction from A to B then, we can solve A as well.
- Suppose we know that A is not solvable (so, colored red), but we don't know how to solve B (so, colored gray). If we find a reduction from A to B then, we can use a proof by contradiction to show that B is not solvable.

The above intuition will manifest differently depending on whether we are studying problems in computability theory (solvable ⇝ decidable) or complexity theory (solvable ⇝ at least as hard).

Theorem 5.1: The following language is undecidable

either accept or reject.

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Proof: We build a proof by contradiction. Suppose for the sake of contradiction that $HALT_{TM}$ is decidable and we will use this assumption to show that L_{TM} is also decidable (which we know is false).

Suppose that R is the decider of $HALT_{TM}$. Then, we use R to build the decider of L_{TM}, namely S.

What is the desired input/output for TM S?

- S is given as an input $\langle M, w\rangle$ (because this is the format of L_{TM}), and it must output "accept" if M accepts w and "reject" if M loops or rejects with input w.
- S cannot simply run M on input w because it is possible that M loops therefore decider S won't halt.
- Instead, S will run the decider (of $HALT_{TM}$) R with input $\langle M, w\rangle$. Now, since R is a decider, we know that R will halt which means that S won't loop.

More formally,

S = "On input $\langle M, w\rangle$, an encoding of TM M and string w:
1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, then S rejects.
3. If R accepts, then $\langle M, w\rangle \in HALT_{TM}$ so it is safe to simulate M on input w.
4. If M accepts, then S accepts. If M rejects, then S rejects."

- In the remaining of the proof, one needs to show that the proposed S is indeed a decider for L_{TM}. ∎

Mapping Reducability (Sipser, Chapter 5.3)

Notice that we have used reductions without formally defining what they do. We will now define mapping reducability, a definition of reducability that uses computable functions.

Definition: A function $f: \Sigma^* \to \Sigma^*$ is a computable function if some Turing machine M, on every input w, halts with just $f(w)$ on its tape

That is, TM M computes f, if $M(w) = f(w)$ for all $w \in \Sigma^*$.

- It is not hard to imagine a machine that takes as an input encodings of numbers and outputs on its tape the result of a mathematical function.

e.g., Input : 2,3 $\xrightarrow{f}$ Output: 2+3=5
then, M's input: 010, 011 $\xrightarrow[\text{M's tape}]{}$ Output: 101

* In the above case, the domain and the range of function f is the set of natural numbers $\mathbb{N}$. What happens if the domain/range represents Turing machine encodings? (recall that a lot of the languages we have seen take as an input $\langle M \rangle$)

e.g., Input: $\langle M_1 \rangle \xrightarrow{f}$ Output: $\langle M_2 \rangle$
then, M's input: $\langle M_1 \rangle \xrightarrow[\text{M's tape}]{}$ Output: $\langle M_2 \rangle$

Thus, the TM M that computes f, takes as an input a machine encoding $\langle M_1 \rangle$ and outputs another machine encoding $\langle M_2 \rangle$.

Definition: Language A is mapping reducible to language B, denoted as $A \leq_m B$, if there is a computable function $f: \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the reduction from A to B.

Theorem 5.22: If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: We let M be the decider of B. Let f be the reduction from A to B. Then, there exists a machine M_f that computes function f. We will construct a decider M' for A using both M and M_f.

M' = "On input w:
1. Run $M_f(w)$ and call the output x.
2. Run M on input x and output whatever M outputs."

If $w \in A$, then from the definition of mapping reducability $f(w) \in B$ holds. Also due to the if-and-only-if condition, if $f(w) \in B$, then $w \in A$. The decider of language B, i.e., machine M, outputs "accept" only when its input is in B. Thus, whenever $M(M_f(w))$ outputs "accept", we have $w \in A$. Therefore, M' is a decider for language A. ■

Corollary 5.23: If $A \leq_m B$ and A is undecidable, then B is undecidable

In Theorem 5.1 we proved that language $HALT_{TM}$ is undecidable.
In the following, we prove it again but this time using mapping reducibility and Corollary 5.23.

Proof Thm 5.1 (Take-2): We have proved that L_{TM} is undecidable.

If we manage to show that $L_{TM} \leq_m HALT_{TM}$, then from Corollary 5.23 we can argue that $HALT_{TM}$ is undecidable as well.

*Notice that this time we do not use a proof by contradiction

To complete the proof, we need to construct a computable function f that takes input of the form $\langle M, w \rangle$ and returns output of the form $\langle M', w \rangle$ such that:

$$\langle M, w \rangle \in L_{TM} \text{ if and only if } \langle M', w \rangle \in HALT_{TM}.$$

The following machine M_f computes a reduction f.

M_f = "On input $\langle M, w \rangle$:

1. Construct the following machine M'

   Notice that in the past we saw high-level descriptions of machines that run/simulate other machines. Here, we have a high-level description of a machine M_f that contains the high-level description of another TM M'.

   M' = "On input x:
   1. Run M on x.
   2. If M accepts, then M' accepts.
   3. If M rejects, then M' enters a loop."

2. Output $\langle M', w \rangle$."

Now we have to argue that the proposed M_f guarantees that:

$$\langle M, w \rangle \in L_{TM} \text{ if and only if } \langle M', w \rangle \in HALT_{TM}.$$

Whenever $\langle M, w \rangle \in L_{TM}$, we know that M accepts w. In this case, given the definition of M', we have that M' accepts w, i.e., $\langle M', w \rangle \in HALT_{TM}$.

Whenever $\langle M', w \rangle \in HALT_{TM}$, we know that M' halts on input w. Given the definition of M', the only way that M' halts is if $M(w)$ accepts. In which case $\langle M, w \rangle \in L_{TM}$. ∎

Theorem 5.28: If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Corollary 5.29: If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.