Foundations of Computing

Lecture 15

Arkady Yerukhimovich

March 9, 2023

Outline

1. Lecture 14 Review
2. Languages on Machines
3. Preliminaries – Countable and Uncountable Sets
4. Proving L_{TM} Undecidable

Lecture 14 Review

- Decidable and Turing-recognizable languages
- Decidability of regular languages

Outline

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepts those in L and rejects those not in L
- Seems to match informal definition we wanted before

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepts those in L and rejects those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Characterizing Computability of Languages

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on ALL inputs, accepts those in L and rejects those not in L
- Seems to match informal definition we wanted before

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Observation

Every Decidable language is also Turing-recognizable, but the reverse direction is not true.

Problems About Regular Languages

Last time, we showed the following languages are decidable:

1. $L_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
2. $L_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$
3. $L_{REX} = \{\langle R, w\rangle \mid R \text{ is a reg. exp. that generates the string } w\}$
4. $L_{\emptyset-DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
5. $L_{EQ-DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 1:

- Every CFG has an equivalent PDA
- Use a TM to run the PDA (easy to simulate stack using TM's tape)

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates } w\}$$

Try 1:

- Every CFG has an equivalent PDA
- Use a TM to run the PDA (easy to simulate stack using TM's tape)

But, there is a problem:

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 1:

- Every CFG has an equivalent PDA
- Use a TM to run the PDA (easy to simulate stack using TM's tape)

But, there is a problem:

- A PDA may have some branches that go on forever – keep pushing and popping things on the stack
- This would mean that on such an input the resulting TM would not halt – i.e., not be a decider

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 2:

- Fortunately, when given a CFG in a certain form (Chomsky Normal Form), can prove that any derivation of $w \in L(G)$ has at most $2n - 1$ steps (where $n = |w|$)

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 2:

- Fortunately, when given a CFG in a certain form (Chomsky Normal Form), can prove that any derivation of $w \in L(G)$ has at most $2n - 1$ steps (where $n = |w|$)
- Moreover, any CFG can be converted into Chomsky Normal Form

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 2:

- Fortunately, when given a CFG in a certain form (Chomsky Normal Form), can prove that any derivation of $w \in L(G)$ has at most $2n - 1$ steps (where $n = |w|$)
- Moreover, any CFG can be converted into Chomsky Normal Form
- Use a TM to list all derivations with $\leq 2n - 1$ steps
  Can do this in finite time, since grammar is finite

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 2:

- Fortunately, when given a CFG in a certain form (Chomsky Normal Form), can prove that any derivation of $w \in L(G)$ has at most $2n - 1$ steps (where $n = |w|$)
- Moreover, any CFG can be converted into Chomsky Normal Form
- Use a TM to list all derivations with $\leq 2n - 1$ steps
  Can do this in finite time, since grammar is finite
- If any of these derivations produce w, accept. Otherwise, reject.

Problems About Context-Free Languages

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

Try 2:

- Fortunately, when given a CFG in a certain form (Chomsky Normal Form), can prove that any derivation of $w \in L(G)$ has at most $2n - 1$ steps (where $n = |w|$)
- Moreover, any CFG can be converted into Chomsky Normal Form
- Use a TM to list all derivations with $\leq 2n - 1$ steps
  Can do this in finite time, since grammar is finite
- If any of these derivations produce w, accept. Otherwise, reject.

Corollary

Every CFL is decidable

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals

$S \to Sa \mid a$

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals
- Idea: For each variable determine if it can be converted to terminals

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals
- Idea: For each variable determine if it can be converted to terminals
- Keep track of which variables can do so, and see if it includes the start variable

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals
- Idea: For each variable determine if it can be converted to terminals
- Keep track of which variables can do so, and see if it includes the start variable

On input $\langle G \rangle$

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals
- Idea: For each variable determine if it can be converted to terminals
- Keep track of which variables can do so, and see if it includes the start variable

On input $\langle G\rangle$

1. Mark all terminals in G

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals
- Idea: For each variable determine if it can be converted to terminals
- Keep track of which variables can do so, and see if it includes the start variable

On input $\langle G\rangle$

1. Mark all terminals in G
2. Repeat until no new variable gets marked:
   - Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, U_2, \ldots, U_k$ has already been marked

Problems About Context-Free Languages

$$L_{\emptyset-CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Intuition:

- Need to test if the start variable can ever generate a string of all terminals
- Idea: For each variable determine if it can be converted to terminals
- Keep track of which variables can do so, and see if it includes the start variable

On input $\langle G\rangle$

1. Mark all terminals in G
2. Repeat until no new variable gets marked:
   - Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, U_2, \ldots, U_k$ has already been marked
3. If starts symbol is not marked, accept. Otherwise, reject

Problems About Context-Free Languages

$$L_{EQ-CFG} = \{\langle G, H\rangle \mid G, H \text{ are CFGs and } L(G) = L(H)\}$$

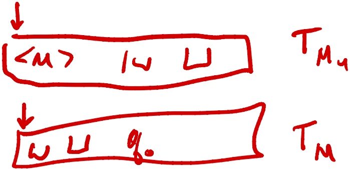
Problems About Context-Free Languages

$$L_{EQ-CFG} = \{\langle G, H\rangle \mid G, H \text{ are CFGs and } L(G) = L(H)\}$$

We will prove after spring break that this is undecidable.

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

$M_u \;(\langle M \rangle)$



Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

Problems About Turing Machines

$$L_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w \rangle$:

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

Is L_{TM} Decidable?:

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

Is L_{TM} Decidable?:

- The problem: M may never halt

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

Is L_{TM} Decidable?:

- The problem: M may never halt
- In this case, above algorithm will never output accept or reject

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

Is L_{TM} Decidable?:

- The problem: M may never halt
- In this case, above algorithm will never output accept or reject
- If could determine that M will never halt (i.e, it has entered an infinite loop), could reject.

Problems About Turing Machines

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Observation: L_{TM} is Turing-recognizable

On input $\langle M, w\rangle$:

1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

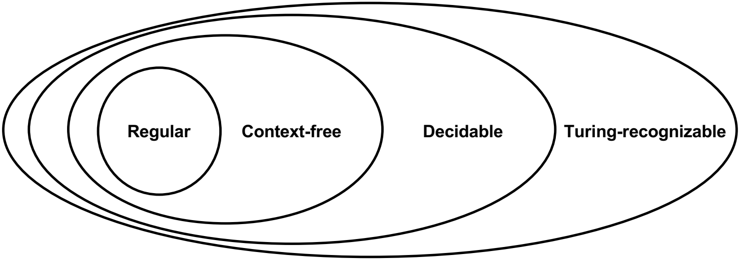
Is L_{TM} Decidable?:

- The problem: M may never halt
- In this case, above algorithm will never output accept or reject
- If could determine that M will never halt (i.e, it has entered an infinite loop), could reject.

An Undecidable Problem

- We will prove that L_{TM} is undecidable

Relationships Among Language Classes



Outline

Set Cardinality

- Cardinality of a set S is the number of elements in that set ($|S|$)

Set Cardinality

- Cardinality of a set S is the number of elements in that set ($|S|$)
- A set S can be finite ($|S| < \infty$) or infinite ($|S| = \infty$)

Set Cardinality

- Cardinality of a set S is the number of elements in that set ($|S|$)
- A set S can be finite ($|S| < \infty$) or infinite ($|S| = \infty$)
- $|S_1| = |S_2|$ if there's a one-to-one and onto mapping from S_1 to S_2

Set Cardinality

- Cardinality of a set S is the number of elements in that set ($|S|$)
- A set S can be finite ($|S| < \infty$) or infinite ($|S| = \infty$)
- $|S_1| = |S_2|$ if there's a one-to-one and onto mapping from S_1 to S_2
- Example:
  $A = \{0, 1, 2, 3\}$
  $B = \{a, b, c, d\}$
  $f(0) = a, f(1) = b, f(2) = c, f(3) = d$

Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

- An infinite set A is *countably infinite* if it has the same cardinality as the natural numbers: $\mathcal{N} = 1, 2, 3, \ldots$

Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

- An infinite set A is *countably infinite* if it has the same cardinality as the natural numbers: $\mathcal{N} = 1, 2, 3, \ldots$
- A set A is countable if it is finite or countably infinite

Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

- An infinite set A is *countably infinite* if it has the same cardinality as the natural numbers: $\mathcal{N} = 1, 2, 3, \ldots$
- A set A is countable if it is finite or countably infinite
- A set that is not countable is *uncountable*

Example 1: Evens

Evens

The set of even numbers is countable

$\mathbb{N} \to$ Evens

$f(x) = 2x$

$f(1) = 2$

$f(2) = 4$

Example 2: Rationals

Rationals

The set of rational numbers is

Example 2: Rationals

Rationals

The set of rational numbers is countable

Example 2: Rationals

Rationals

The set of rational numbers is countable

	1	2	3	$\cdots$
1	1/1	1/2	1/3	
2	2/1	2/2	2/3	$\cdots$
3	3/1	3/2	3/3	
4	4/1	4/2	$\vdots$	

Example 3: Strings

Strings

The set of strings in $\{0,1\}^*$ is countable

$\epsilon, 0, 1, 00, 01, 10, 11, \ldots$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

n	f(n)
1	1.234...
2	3.141...
3	5.556...
$\vdots$	$\vdots$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

n	f(n)
1	1.234...
2	3.141...
3	5.556...
⋮	⋮

- We construct a value $x \in \mathcal{R}$ s.t $x \neq f(n)$ for any n
  Idea: For all $i \in \mathcal{N}$, make $x_i \neq f(i)_i$ $x = 0.314...$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

n	f(n)
1	1.234...
2	3.141...
3	5.556...
⋮	⋮

- We construct a value $x \in \mathcal{R}$ s.t $x \neq f(n)$ for any n
  Idea: For all $i \in \mathcal{N}$, make $x_i \neq f(i)_i$
- Contradiction – f is not mapping between $\mathcal{R}$ and $\mathcal{N}$

Outline

The Set of Turing Machines

Turing Machines

The set of Turing Machines is countable

The Set of Turing Machines

Turing Machines

The set of Turing Machines is countable

- We already showed that the set of strings $\{0,1\}^*$ is countable

The Set of Turing Machines

Turing Machines

The set of Turing Machines is countable

- We already showed that the set of strings $\{0,1\}^*$ is countable
- Can similarly show that for any finite alphabet Σ, Σ^* is countable

The Set of Turing Machines

Turing Machines

The set of Turing Machines is countable

- We already showed that the set of strings $\{0,1\}^*$ is countable
- Can similarly show that for any finite alphabet Σ, Σ^* is countable
- But, a TM M can be written as a string $\langle M \rangle \in \Sigma^*$

The Set of Turing Machines

Turing Machines

The set of Turing Machines is countable

- We already showed that the set of strings $\{0, 1\}^*$ is countable
- Can similarly show that for any finite alphabet Σ, Σ^* is countable
- But, a TM M can be written as a string $\langle M \rangle \in \Sigma^*$
- Hence, by omitting all strings that are not encodings of valid TMs we get a mapping of TMs to $\mathcal{N}$

The Set of Languages

Languages over alphabet Σ

The set of all languages ($\mathcal{L}$) over alphabet Σ is uncountable

The Set of Languages

Languages over alphabet Σ

The set of all languages ($\mathcal{L}$) over alphabet Σ is uncountable

Proof:

1. The set B of infinite binary sequences
   - An infinite binary sequence is an infinite length strings of 0's and 1's
   - B is uncountable – can prove similar to $\mathcal{R}$

$B = \{ 0101100\ldots,\ 1100\ldots,\ \}$

The Set of Languages

Languages over alphabet Σ

The set of all languages ($\mathcal{L}$) over alphabet Σ is uncountable

Proof:

1. The set B of infinite binary sequences
   - An infinite binary sequence is an infinite length strings of 0's and 1's
   - B is uncountable – can prove similar to $\mathcal{R}$
2. Set of languages ($\mathcal{L}$) has same cardinality as B

The Set of Languages

Languages over alphabet Σ

The set of all languages ($\mathcal{L}$) over alphabet Σ is uncountable

Proof:

1. The set B of infinite binary sequences
   - An infinite binary sequence is an infinite length strings of 0's and 1's
   - B is uncountable – can prove similar to $\mathcal{R}$
2. Set of languages ($\mathcal{L}$) has same cardinality as B
   - Define the characteristic sequence χ_A of language $A \in \mathcal{L}$

$$\begin{array}{ccccccccccc}
\Sigma^* & = & \{ & \epsilon & 0 & 1 & 00 & 01 & 11 & 000 & \cdots & \} \\
A & = & \{ & & & 1 & 00 & & & 000 & \cdots & \} \\
\chi_A & = & & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots &
\end{array}$$

Languages over alphabet Σ

The set of all languages ($\mathcal{L}$) over alphabet Σ is uncountable

Proof:

1. The set B of infinite binary sequences
   - An infinite binary sequence is an infinite length strings of 0's and 1's
   - B is uncountable – can prove similar to $\mathcal{R}$
2. Set of languages ($\mathcal{L}$) has same cardinality as B
   - Define the characteristic sequence χ_A of language $A \in \mathcal{L}$

$$\begin{array}{ccccccccccc}
\Sigma^* & = & \{ & \epsilon & 0 & 1 & 00 & 01 & 11 & 000 & \cdots & \} \\
A & = & \{ & & & 1 & 00 & & & 000 & \cdots & \} \\
\chi_A & = & & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots &
\end{array}$$

   - This is a one-to-one and onto mapping from $\mathcal{L}$ to B, so $|\mathcal{L}| = |B|$

The Set of Languages

Languages over alphabet Σ

The set of all languages ($\mathcal{L}$) over alphabet Σ is uncountable

Proof:

1. The set B of infinite binary sequences
   - An infinite binary sequence is an infinite length strings of 0's and 1's
   - B is uncountable – can prove similar to $\mathcal{R}$
2. Set of languages ($\mathcal{L}$) has same cardinality as B
   - Define the characteristic sequence χ_A of language $A \in \mathcal{L}$

$$\begin{array}{ccccccccccc} \Sigma^* & = & \{ & \epsilon & 0 & 1 & 00 & 01 & 11 & 000 & \cdots & \} \\ A & = & \{ & & & 1 & 00 & & & 000 & \cdots & \} \\ \chi_A & = & & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & \end{array}$$

   - This is a one-to-one and onto mapping from $\mathcal{L}$ to B, so $|\mathcal{L}| = |B|$
3. Therefore, $\mathcal{L}$ is uncountable

Some Languages are not Turing-recognizable

We have proven:

1. The set of Turing Machines is countable
2. The set of languages is uncountable

Some Languages are not Turing-recognizable

We have proven:

1. The set of Turing Machines is countable
2. The set of languages is uncountable

Therefore:

Some Languages are not Turing-recognizable

We have proven:

1. The set of Turing Machines is countable
2. The set of languages is uncountable

Therefore:

- There is no one-to-one and onto mapping from languages to Turing Machines

Some Languages are not Turing-recognizable

We have proven:

1. The set of Turing Machines is countable
2. The set of languages is uncountable

Therefore:

- There is no one-to-one and onto mapping from languages to Turing Machines
- Thus, there exist languages that have no corresponding TM that recognizes them

Some Languages are not Turing-recognizable

We have proven:

1. The set of Turing Machines is countable
2. The set of languages is uncountable

Therefore:

- There is no one-to-one and onto mapping from languages to Turing Machines
- Thus, there exist languages that have no corresponding TM that recognizes them
- Note, that such languages are also undecidable

Where are we now?

- We have proven that some languages are not Turing-recognizable

Some Languages are not Turing-recognizable

We have proven:

1. The set of Turing Machines is countable
2. The set of languages is uncountable

Therefore:

- There is no one-to-one and onto mapping from languages to Turing Machines
- Thus, there exist languages that have no corresponding TM that recognizes them
- Note, that such languages are also undecidable

Where are we now?

- We have proven that some languages are not Turing-recognizable
- But, we have not given any examples of such a language

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that L_{TM} is decided by TM H

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

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that L_{TM} is decided by TM H

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

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that L_{TM} is decided by TM H

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

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:
  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

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that L_{TM} is decided by TM H

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

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:
  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

$$D(\langle M\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

L_{TM} is Undecidable

$$L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that L_{TM} is decided by TM H

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

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:
  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

$$D(\langle M\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

- Now consider what happens if we run D on $\langle D\rangle$

$$D(\langle D\rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D\rangle \end{cases}$$

How Is This a Diagonalization?

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	$\underline{accept}$	$reject$	$accept$		$accept$	
M_2	$reject$	$\underline{reject}$	$reject$	$\ldots$	$accept$	$\ldots$
M_3	$accept$	$accept$	$\underline{accept}$		$reject$	
$\vdots$		$\vdots$		$\ddots$		
D	$reject$	$accept$	$reject$		?	

- We have defined D to do the opposite of what M_i does on input $\langle M_i \rangle$
- But what does D do on input $\langle D \rangle$??

A Turing-unrecognizable Language

$\overline{L_{TM}}$

The language

$$\overline{L_{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) \neq 1\}$$

is not Turing-recognizable