Foundations of Computing

Lecture 14

Arkady Yerukhimovich

March 4, 2025

Outline

1. Lecture 13 Review
2. Decidable and Turing-recognizable Languages
3. Languages With Machines as Input
4. Preliminaries – Countable and Uncountable Sets

Lecture 13 Review

- More Turing Machines
- Turing Machine Variants
  - Multi-tape Turing Machines
  - Non-deterministic Turing Machines

Outline

1. Lecture 13 Review
2. Decidable and Turing-recognizable Languages
3. Languages With Machines as Input
4. Preliminaries – Countable and Uncountable Sets

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L
- That is, given a string w, an algorithm (aka. a Turing Machine) should tell us whether $w \in L$ or not.

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L
- That is, given a string w, an algorithm (aka. a Turing Machine) should tell us whether $w \in L$ or not.

Question(s)

- Can all languages be computed in this way?

What Does It Mean to Compute?

- We have modeled computation as recognizing a language L
- That is, given a string w, an algorithm (aka. a Turing Machine) should tell us whether $w \in L$ or not.

Question(s)

- Can all languages be computed in this way?
- Are there some problems that inherently do not have any algorithmic solution?

Characterizing Computability of Languages

Definition: Decidable languages

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

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Definition: Turing-recognizable languages

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* if some TM M recognizes strings in L

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* if some TM M recognizes strings in L

- M halts and accepts on all strings $w \in L$

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* if some TM M recognizes strings in L

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

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* if some TM M recognizes strings in L

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

Observations:

- L is Turing-recognizable if can recognize yes instances, L is decidable if can also recognize no instances.

Characterizing Computability of Languages

Definition: Decidable languages

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

- M halts on ALL inputs, accepting all $w \in L$ and rejecting all $w \notin L$

Definition: Turing-recognizable languages

A language L is *Turing-recognizable* if some TM M recognizes strings in L

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

Observations:

- L is Turing-recognizable if can recognize yes instances, L is decidable if can also recognize no instances.
- Every Decidable language is also Turing-recognizable, but the reverse direction may not be true.

The Big Question

The Question

Are there problems that are undecidable?

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm
- If you believe Church-Turing thesis, it cannot be solved by any computer

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm
- If you believe Church-Turing thesis, it cannot be solved by any computer
- We will see that even relatively natural problems can be undecidable

The Big Question

The Question

Are there problems that are undecidable?

What would this mean:

- Such a problem is not solvable by any algorithm
- If you believe Church-Turing thesis, it cannot be solved by any computer
- We will see that even relatively natural problems can be undecidable

A Second Question

What about Turing-unrecognizable languages?

Outline

Taking Machines as Input

- Recall that we have defined machines as tuples:

Taking Machines as Input

- Recall that we have defined machines as tuples:
    1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
    2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
    3. TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- This means that any such machine can be written down as a finite length string

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- This means that any such machine can be written down as a finite length string
- So, can give a description of a machine M to another machine M'

Taking Machines as Input

- Recall that we have defined machines as tuples:
  1. DFA/NFA $M = (Q, \Sigma, \delta, q1, F)$
  2. PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  3. TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
- This means that any such machine can be written down as a finite length string
- So, can give a description of a machine M to another machine M'
- We already talked about a universal TM which can run any other TM

Problems About Regular Languages

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Problems About Regular Languages

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Algorithm to decide A_{DFA}:
On input $\langle B, w\rangle$

Problems About Regular Languages

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Algorithm to decide A_{DFA}:
On input $\langle B, w\rangle$

1. Simulate B on input w

Problems About Regular Languages

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Algorithm to decide A_{DFA}:
On input $\langle B, w\rangle$

1. Simulate B on input w
2. If simulation ends in an accept, then accept. If it ends in a non-accepting state, then reject

Problems About Regular Languages

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Problems About Regular Languages

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Algorithm to decide A_{NFA}:
On input $\langle B, w\rangle$

Problems About Regular Languages

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Algorithm to decide A_{NFA}:
On input $\langle B, w\rangle$

1. Convert NFA B to equivalent DFA C

Problems About Regular Languages

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Algorithm to decide A_{NFA}:
On input $\langle B, w\rangle$

1. Convert NFA B to equivalent DFA C
2. Run TM from previous slide on input $\langle C, w\rangle$
3. Output what this TM outputs

Problems About Regular Languages

$$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a reg. exp. that generates the string } w\}$$

Problems About Regular Languages

$$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a reg. exp. that generates the string } w\}$$

Algorithm to decide A_{REX}:
On input $\langle R, w\rangle$

1. Convert R to equivalent DFA M
2. Use TM from previous slide to decide it

Problems About Regular Languages

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } \underline{L(A) = \emptyset}\}$$

Problems About Regular Languages

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Intuition:

- Remember, for a DFA to accept some string there must be a path from the start state to an accept state

Problems About Regular Languages

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Intuition:

- Remember, for a DFA to accept some string there must be a path from the start state to an accept state
- We need to figure out if such a path exists

Problems About Regular Languages

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Intuition:

- Remember, for a DFA to accept some string there must be a path from the start state to an accept state
- We need to figure out if such a path exists

Algorithm to decide E_{DFA}:
On input $\langle A \rangle$

Problems About Regular Languages

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Intuition:

- Remember, for a DFA to accept some string there must be a path from the start state to an accept state
- We need to figure out if such a path exists

Algorithm to decide E_{DFA}:
On input $\langle A \rangle$

1. Mark the start state of A

Problems About Regular Languages

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Intuition:

- Remember, for a DFA to accept some string there must be a path from the start state to an accept state
- We need to figure out if such a path exists

Algorithm to decide E_{DFA}:
On input $\langle A \rangle$

1. Mark the start state of A
2. Repeat until no new states get marked:
   - Mark any state that has an incoming transition from any state already marked

Problems About Regular Languages

$$E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Intuition:

- Remember, for a DFA to accept some string there must be a path from the start state to an accept state
- We need to figure out if such a path exists

Algorithm to decide E_{DFA}:
On input $\langle A\rangle$

1. Mark the start state of A
2. Repeat until no new states get marked:
   - Mark any state that has an incoming transition from any state already marked
3. If no accept state is marked, accept, else, reject

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C \rangle$ to decide this language

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B \rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C \rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in L(A)$ s.t. $x \notin L(B)$

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in \underline{L(A)}$ s.t. $x \notin L(B)$
  $L(A) \setminus L(B) = L(A) \cap \overline{L(B)}$ contains all such items

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in L(A)$ s.t. $x \notin L(B)$
  $L(A) \setminus L(B) = L(A) \cap \overline{L(B)}$ contains all such items
- Need to also consider items in $L(B)$ that are not in $L(A)$

Problems About Regular Languages

$$EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$$

Intuition:

- Construct regular language that is empty if and only if $L(A) = L(B)$
- Since this language is regular, there is a DFA C that decides it
- Use TM from previous example on input $\langle C\rangle$ to decide this language

Constructing $L(C)$:

- We need to find all the differences between $L(A)$ and $L(B)$
- Consider all items $x \in L(A)$ s.t. $x \notin L(B)$
  $L(A) \setminus L(B) = L(A) \cap \overline{L(B)}$ contains all such items
- Need to also consider items in $L(B)$ that are not in $L(A)$

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

Problems About Context-Free Languages

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

Problems About Context-Free Languages

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

Intuition:

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

Problems About Context-Free Languages

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

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

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

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

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

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

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

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

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

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

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

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 Turing Machines

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

Problems About Turing Machines

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

Q1: Is A_{TM} Turing-recognizable?

Problems About Turing Machines

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

Q1: Is A_{TM} Turing-recognizable?

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

Problems About Turing Machines

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

Q1: Is A_{TM} Turing-recognizable?

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

1. Simulate M on input w

Problems About Turing Machines

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

Q1: Is A_{TM} 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

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

Q1: Is A_{TM} 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

Since $M(w) = 1$ iff M halts and accepts, the above algorithm will halt and accept all yes instances.

Problems About Turing Machines

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

Q1: Is A_{TM} 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

Since $M(w) = 1$ iff M halts and accepts, the above algorithm will halt and accept all yes instances.

Q2: Is A_{TM} decidable?

Problems About Turing Machines

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

Q1: Is A_{TM} 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

Since $M(w) = 1$ iff M halts and accepts, the above algorithm will halt and accept all yes instances.

Q2: Is A_{TM} decidable?

- M may never halt on some w

Problems About Turing Machines

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

Q1: Is A_{TM} 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

Since $M(w) = 1$ iff M halts and accepts, the above algorithm will halt and accept all yes instances.

Q2: Is A_{TM} decidable?

- M may never halt on some w
- In this case, above algorithm will never output accept or reject

Problems About Turing Machines

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

Q1: Is A_{TM} 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

Since $M(w) = 1$ iff M halts and accepts, the above algorithm will halt and accept all yes instances.

Q2: Is A_{TM} decidable?

- M may never halt on some w
- 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

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

Q1: Is A_{TM} Turing-recognizable?

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

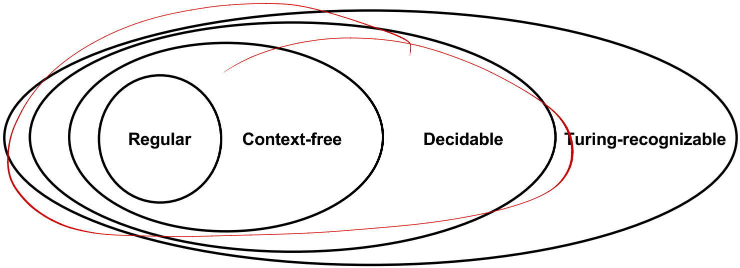
1. Simulate M on input w
2. If M ever enters its accept state, halt and accept. If M ever enters its reject state, halt and reject

Since $M(w) = 1$ iff M halts and accepts, the above algorithm will halt and accept all yes instances.

Q2: Is A_{TM} decidable?

- M may never halt on some w
- 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.
- But, how do we determine this?

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:
  $A = \{0, 1, 2, 3\}$
  $B = \{a, b, c, d\}$
  $f(0) = a, f(1) = b, f(2) = c, f(3) = d$

$$\begin{matrix} 1 & 2 & 3 & 4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 4 & 6 & 8 \end{matrix}$$

Countable and Uncountable Sets

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

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

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

Diagonalization

Real Numbers

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

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

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 $\mathcal{R}$

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...
$\vdots$	$\vdots$

- 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 $x_i \neq f(i)_i$

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 $\mathcal{N}$