Foundations of Computing

Lecture 26 – Final Exam Review

Arkady Yerukhimovich

April 27, 2023

Outline

Lecture 25 Review

- Zero-Knowledge Proofs
- Where's Waldo
- Puppy and Panda
- Graph Isomorphism

We Are Done!

Welcome to the last lecture of CS 3313!!!

- Complete course evaluation form for 5 points on final exam

- Get a cookie

Outline

Complexity Classes

Complexity Classes

- $\mathcal{P}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$
- co-$\mathcal{NP}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$
- co-$\mathcal{NP}$
- $\mathcal{IP}$

Complexity Classes

- $\mathcal{P}$
- $\mathcal{NP}$
- co-$\mathcal{NP}$
- $\mathcal{IP}$

Important

Make sure you know the definitions and relationships between these complexity classes.

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

- $f(n) = O(n^3)$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

- $f(n) = O(n^3)$
- For every $n \geq 6$, $f(n) \leq 6n^3$
- I.e., $n_0 = 6, c = 6$

Asymptotic Notation – Big-O

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f(n) = O(g(n))$ if

- There exist positive integers c, n_0 s.t. for all $n \geq n_0$

$$f(n) \leq cg(n)$$

Example

$$f(n) = 5n^3 + 3n^2 + 10n + 8$$

- $f(n) = O(n^3)$
- For every $n \geq 6$, $f(n) \leq 6n^3$
- I.e., $n_0 = 6, c = 6$
- Note that $f(n) = O(n^4)$

Outline

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

- $\mathcal{P}$ corresponds to the class of "efficiently-solvable" problems

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

- $\mathcal{P}$ corresponds to the class of "efficiently-solvable" problems
- $\mathcal{P}$ is invariant for all models of computation polynomially-equivalent to 1-tape TM

Complexity Class $\mathcal{P}$

Definition

$\mathcal{P}$ is the class of languages decidable in polynomial time on a 1-tape deterministic TM.

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

- $\mathcal{P}$ corresponds to the class of "efficiently-solvable" problems
- $\mathcal{P}$ is invariant for all models of computation polynomially-equivalent to 1-tape TM
- $\mathcal{P}$ has nice closure properties

Problems in $\mathcal{P}$

- PATH
- RELPRIME
- Anything you saw in algorithms class

Outline

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time
- $\mathcal{NP}$ is the class of problems where you can verify a solution in poly-time

The Class $\mathcal{NP}$

Definition

$\mathcal{NP}$ is the class of languages that have polynomial time verifiers.

- We already saw that *HAMPATH* and *SAT* are in $\mathcal{NP}$
- Every $L \in \mathcal{P}$ is also in $\mathcal{NP}$: $\mathcal{P} \subseteq \mathcal{NP}$

Intuition

- $\mathcal{P}$ is the class of problems where you can find a solution in poly-time
- $\mathcal{NP}$ is the class of problems where you can verify a solution in poly-time
- Question: $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent – A language L is poly-time verifiable if and only if it is decided by a poly-time NTM.

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent – A language L is poly-time verifiable if and only if it is decided by a poly-time NTM.

Proof Idea:

- Need to prove both directions
- An NTM simulates the verifier by guessing the witness w

The Class $\mathcal{NP}$ – Another Formulation

- $\mathcal{NP}$ stands for non-deterministic polynomial time
- $\mathcal{NP}$ is the set of languages decided by poly-time NTMs

Theorem

The two definitions of $\mathcal{NP}$ are equivalent – A language L is poly-time verifiable if and only if it is decided by a poly-time NTM.

Proof Idea:

- Need to prove both directions
- An NTM simulates the verifier by guessing the witness w
- A verifier simulates the NTM by using the accepting branch as the witness

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ there exists a witness w s.t. $V(x, w) = 1$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ there exists a witness w s.t. $V(x, w) = 1$

co-$\mathcal{NP}$

$L \in$ co-$\mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ for all w, $V(x, w) = 0$

$\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

$\mathcal{P}$

$L \in \mathcal{P}$ if there exists poly-time DTM M s.t $M(x) = [x \in L]$

$\mathcal{NP}$

$L \in \mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ there exists a witness w s.t. $V(x, w) = 1$

co-$\mathcal{NP}$

$L \in$ co-$\mathcal{NP}$ if there exists poly-time DTM V s.t. for $x \in L$ for all w, $V(x, w) = 0$

Question:

$$\text{Is } \mathcal{NP} = \text{co-}\mathcal{NP}?$$

Problems in $\mathcal{NP}$

- CLIQUE

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT
- Vertex cover

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT
- Vertex cover
- Independent set

Problems in $\mathcal{NP}$

- CLIQUE
- Subset Sum
- Graph isomorphism
- Graph Hamiltonicity
- Satisfiability
- 3-SAT
- Vertex cover
- Independent set
- and many more

Important

Make sure you know how to prove $L \in \mathcal{NP}$

Outline

Mapping Reductions

Mapping Reduction

Language A is mapping reducible to language B ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

$$x \in A \iff f(x) \in B$$

Mapping Reductions

Mapping Reduction

Language A is mapping reducible to language B ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

$$x \in A \iff f(x) \in B$$

Poly-time Mapping Reduction

Language A is poly-time mapping reducible to language B ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

$$x \in A \iff f(x) \in B$$

Mapping Reductions

Mapping Reduction

Language A is mapping reducible to language B ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

$$x \in A \iff f(x) \in B$$

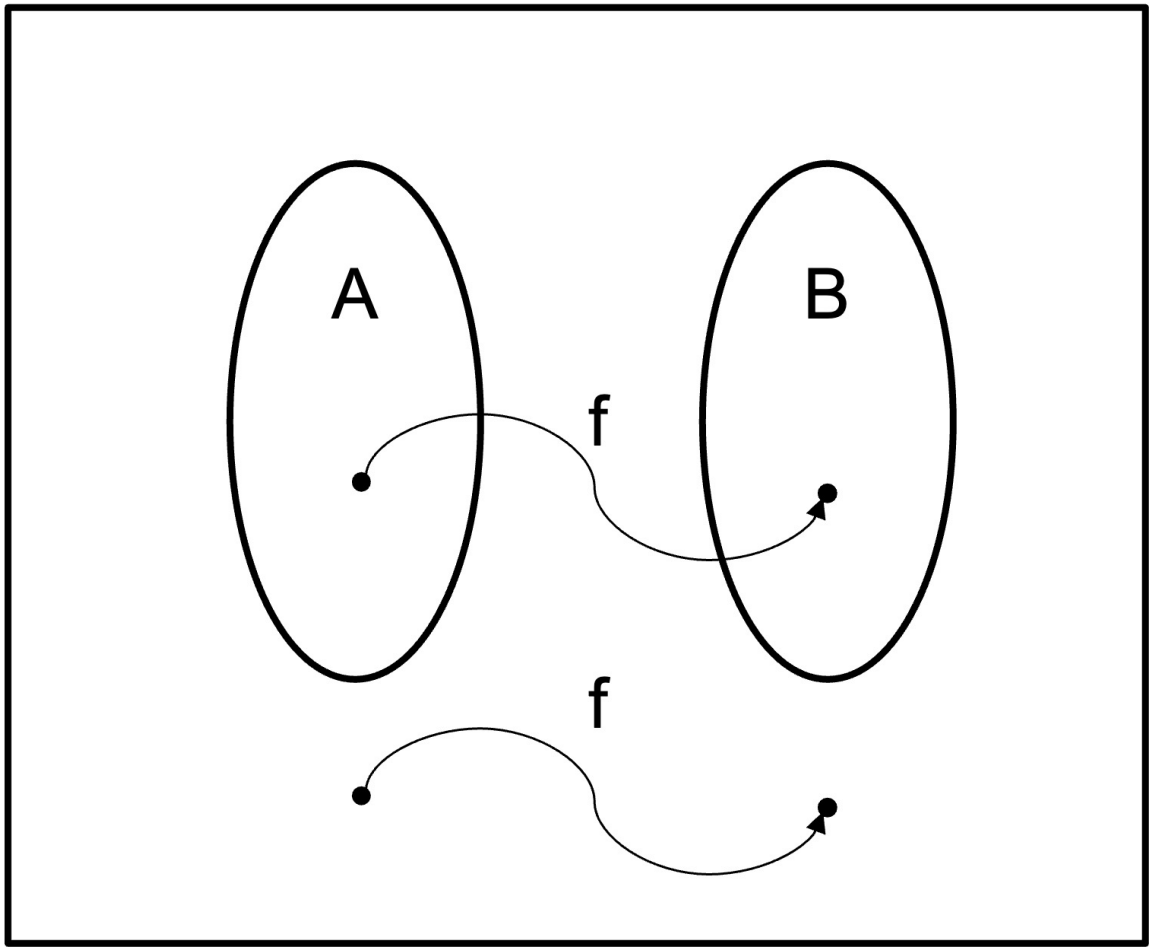
Poly-time Mapping Reduction

Language A is poly-time mapping reducible to language B ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$, where for every x,

$$x \in A \iff f(x) \in B$$

- Poly-time reductions give an efficient way to convert membership testing in A to membership testing in B
- If B has a poly-time solution so does A

Poly-time Mapping Reductions



f runs in time $poly(|x|)$ on all inputs x

Why Poly-Time Reductions

Theorem

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let M be the poly-time TM deciding B
- Let f be the poly-time reduction from A to B
- Can construct M' deciding A:
  $M' =$ On input x:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever M outputs

Why Poly-Time Reductions

Theorem

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let M be the poly-time TM deciding B
- Let f be the poly-time reduction from A to B
- Can construct M' deciding A:
  $M' =$ On input x:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever M outputs
     - If $x \in A$, $f(x) \in B$ so M accepts
     - If $x \notin A$, $f(x) \notin B$, so M rejects
     - Since both f and M are poly-time, $M(f(x))$ is also poly-time

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

- If ϕ is satisfiable then G has a k-clique

3SAT $\leq_P$ CLIQUE

- Need to show reduction f from 3SAT formula ϕ to $\langle G, k \rangle$ where
  - If ϕ is satisfiable, G has a clique of size k
  - If ϕ is not satisfiable, G has no clique of size k
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

- If ϕ is satisfiable then G has a k-clique
- If G has a k-clique then ϕ is satisfiable

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

Theorem

If B is $\mathcal{NP}$-complete and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

$\mathcal{NP}$-Completeness

Definition

A language B is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- B is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

Theorem

If B is $\mathcal{NP}$-complete and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

Theorem

If B is $\mathcal{NP}$-complete and $B \leq_P C$ for $C \in \mathcal{NP}$, then C is $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ NAE-4SAT

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ NAE-4SAT
7. NAE-4SAT $\leftarrow_P$ 3-Coloring

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ NAE-4SAT
7. NAE-4SAT $\leftarrow_P$ 3-Coloring
8. More on the HW

$\mathcal{NP}$-Complete Languages

1. SAT is $\mathcal{NP}$-complete
2. 3-SAT is $\mathcal{NP}$-complete
3. 3-SAT $\leq_P$ CLIQUE – So CLIQUE in $\mathcal{NP}$-complete
4. 3-SAT $\leq_P$ Vertex Cover
5. Vertex Cover $\leq_P$ Independent Set
6. 3-SAT $\leq_P$ NAE-4SAT
7. NAE-4SAT $\leftarrow_P$ 3-Coloring
8. More on the HW

Important

Make sure you remember what direction the reduction should go.

Outline

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

1. (Completeness) If $x \in L$, then $\Pr[\langle P, V \rangle(x) = 1] = 1$

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

1. (Completeness) If $x \in L$, then $\Pr[\langle P, V \rangle(x) = 1] = 1$
2. (Soundness) If $x \notin L$, then for any (possibly unbounded) P^*, we have $\Pr[\langle P^*, V \rangle(x) = 1] \leq 1/2$

The Class $\mathcal{IP}$

Definition

$L \in \mathcal{IP}$ if there exist a pair of interactive algorithms (P, V) with V being poly-time (in $|x|$) s.t.

1. (Completeness) If $x \in L$, then $\Pr[\langle P, V \rangle(x) = 1] = 1$
2. (Soundness) If $x \notin L$, then for any (possibly unbounded) P^*, we have $\Pr[\langle P^*, V \rangle(x) = 1] \leq 1/2$

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0, 1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic
   - Then G^* is isomorphic to both G_0 and G_1

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic
   - Then G^* is isomorphic to both G_0 and G_1
   - P has no way to tell which one V started from

Graph Non-Isomorphism

Question

How can we prove that two graphs G_0 and G_1 are NOT isomorphic?

The Protocol:

1. V chooses $b \leftarrow \{0,1\}$, and applies a random permutation π to the vertices of G_b and sends this graph G^* to P
2. P determines if G^* is isomorphic to G_0 and sends $b' = 0$ if so, or $b' = 1$ otherwise back to V
3. V accepts if $b' = b$

Why This Works:

1. (Completeness) Suppose that G_0 and G_1 are not isomorphic.
   - Then G^* can only be isomorphic to one of the two graphs
   - P can perfectly determine which one this is
   - So $\Pr[b' = b] = 1$
2. (Soundness) Suppose that G_0 and G_1 are isomorphic
   - Then G^* is isomorphic to both G_0 and G_1
   - P has no way to tell which one V started from
   - Thus, $\Pr[b' = b] = 1/2$

Another Example – Polynomial Identity Testing

PIT Problem

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Question: What should V do?

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Question: What should V do?

- Suppose that V is deterministic:

Another Example – Polynomial Identity Testing

PIT Problem

- Prover P has a degree d polynomial f and wants to prove that

$$\forall x, f(x) = 0$$

- V is allowed to query $f(x)$ at points x of its choice – but, P knows V's strategy

Question: What should V do?

- Suppose that V is deterministic:
- What if you allow V to be randomized:

Arithmetization

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Constructing Φ:

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Constructing Φ:

- Identify 0 =false, and positive integer = true

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Constructing Φ:

- Identify 0 =false, and positive integer = true
- $x_i \to x_i$, $\overline{x_i} \to (1 - x_i)$

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Constructing Φ:

- Identify 0 =false, and positive integer = true
- $x_i \to x_i$, $\overline{x_i} \to (1 - x_i)$
- Replace $\vee$ with $+$

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Constructing Φ:

- Identify 0 =false, and positive integer = true
- $x_i \to x_i$, $\overline{x_i} \to (1 - x_i)$
- Replace $\vee$ with $+$
- Replace $\wedge$ with $\times$

Arithmetization

Idea

For n inputs $x_1, x_2, \ldots, x_n$, construct a polynomial $\Phi(x_1, \ldots, x_n)$ s.t.

$$\Phi(x_1, \ldots, x_n) = 0 \text{ if and only if } \phi(x_1, \ldots, x_n) \text{ is unsatisfiable}$$

Constructing Φ:

- Identify 0 =false, and positive integer = true
- $x_i \to x_i$, $\overline{x_i} \to (1 - x_i)$
- Replace $\vee$ with $+$
- Replace $\wedge$ with $\times$

$$\phi(x) = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

Remember

Arithmetization is key component of proof that co-$\mathcal{NP} \subseteq \mathcal{IP}$

Languages in $\mathcal{IP}$

- $\mathcal{P} \subseteq \mathcal{IP}$
- $\mathcal{NP} \subseteq \mathcal{IP}$
- co-$\mathcal{NP} \subseteq \mathcal{IP}$
- PSPACE$=\mathcal{IP}$

$\neq$

$P \subsetneq NP \subsetneq PSPACE$

Outline

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proof

A proof $\langle P, V \rangle(x)$ for a language L is *zero-knowledge* if

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proof

A proof $\langle P, V \rangle(x)$ for a language L is *zero-knowledge* if

- For any (possibly malicious) poly-time verifier V^*

Zero-Knowledge Proofs

Consider an interactive proof between Prover (P) and Verifier (V):

$$\langle P, V \rangle(x)$$

Define V's view of this interaction by:

$$VIEW_V(\langle P, V \rangle(x))$$

This includes:

- V's randomness
- Any messages that V receives

Zero-Knowledge Proof

A proof $\langle P, V \rangle(x)$ for a language L is *zero-knowledge* if

- For any (possibly malicious) poly-time verifier V^*
- There exists a poly-time *Simulator* S s.t.

$$\forall x \in L, \qquad VIEW_{V^*}(\langle P, V^* \rangle(x)) = S(x)$$

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0, 1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0, 1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V
2. V chooses $b' \leftarrow \{0, 1\}$ and sends it to P

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0, 1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V
2. V chooses $b' \leftarrow \{0, 1\}$ and sends it to P
3. P sends V the permutation π' mapping $G_{b'}$ to H

$$\pi' = \begin{cases} \sigma & \text{if } b = b' \\ \sigma\pi^{-1} & \text{if } b = 0, b' = 1 \\ \sigma\pi & \text{if } b = 1, b' = 0 \end{cases}$$

Graph Isomorphism

Input: $x = (G_0, G_1)$
Prover's goal: Prove that he knows permutation π s.t. $\pi(G_0) = G_1$

The Proof

1. P chooses $b \leftarrow \{0, 1\}$ and a random permutation σ and sends $H = \sigma(G_b)$ to V
2. V chooses $b' \leftarrow \{0, 1\}$ and sends it to P
3. P sends V the permutation π' mapping $G_{b'}$ to H

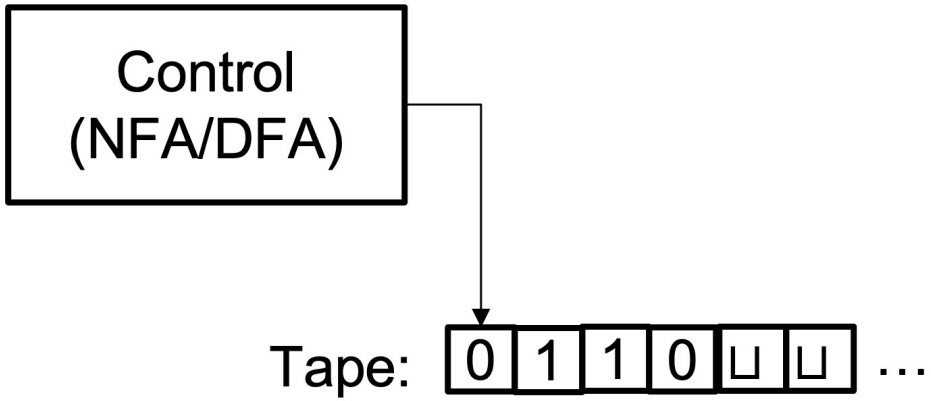
$$\pi' = \begin{cases} \sigma & \text{if } b = b' \\ \sigma\pi^{-1} & \text{if } b = 0, b' = 1 \\ \sigma\pi & \text{if } b = 1, b' = 0 \end{cases}$$

4. V accepts iff $H = \pi'(G_{b'})$

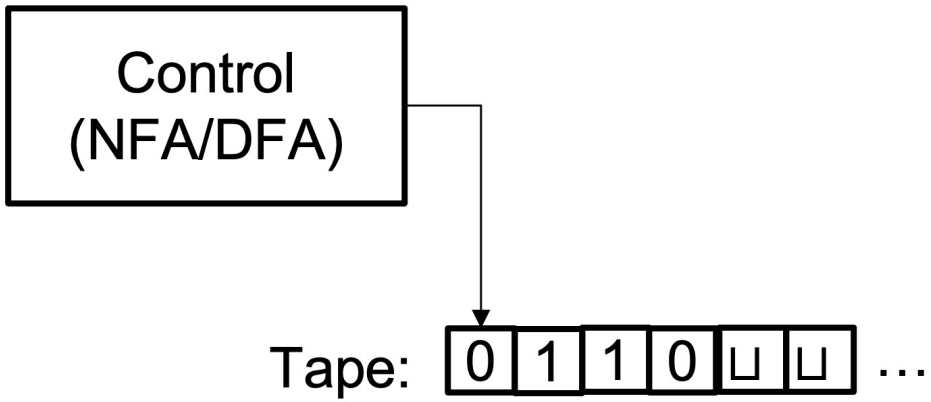
Outline

Outline

The Turing Machine



The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start
- The memory tape is infinite
- Control FA has accept and reject states that are immediately output if entered

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis
- To prove that some problem cannot be solved by an algorithm, enough to reason about Turing Machines
- This means that Turing Machines give an abstraction to capture "feasible computation"

Outline

Characterizing Computability of Languages

Characterizing Computability of Languages

Definition: Recursively enumerable languages

Characterizing Computability of Languages

Definition: Recursively enumerable languages

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

Characterizing Computability of Languages

Definition: Recursively enumerable languages

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

- M halts and accepts all strings in L

Characterizing Computability of Languages

Definition: Recursively enumerable 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: Recursively enumerable 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

Definition: Decidable languages

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

Characterizing Computability of Languages

Definition: Recursively enumerable 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

Definition: Decidable languages

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

- M halts on all inputs, accepting those in L and rejecting those not in L

Take Away

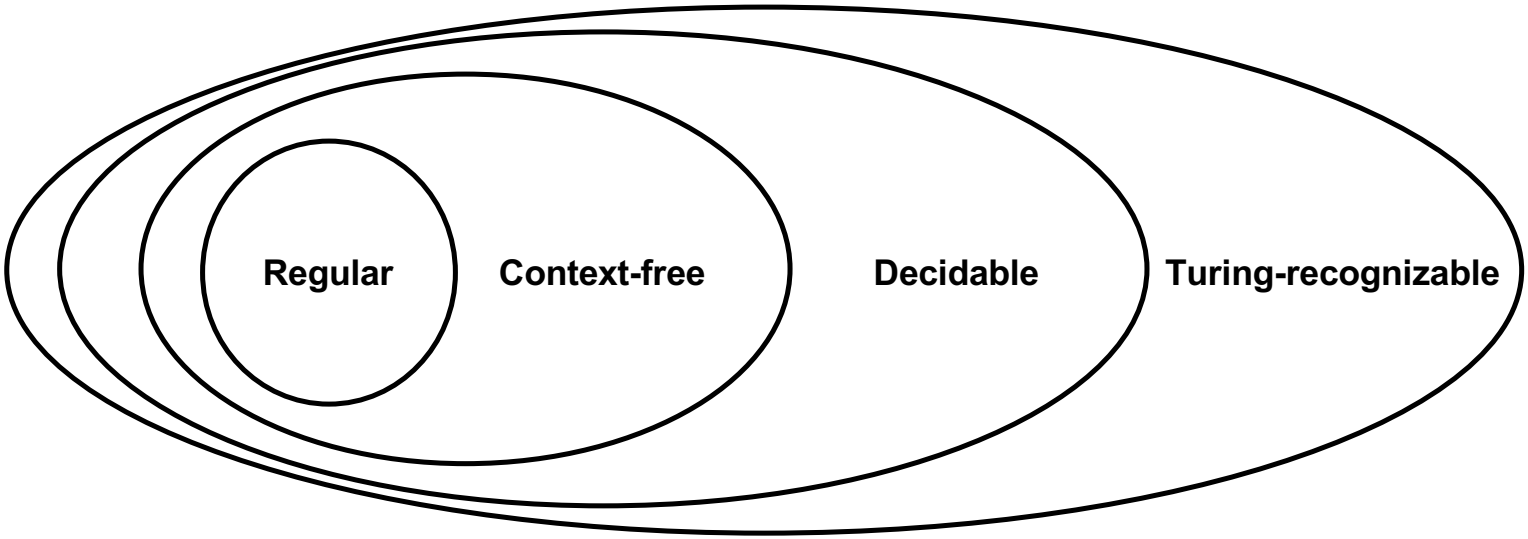
You should be able to show that a language is decidable or Turing-recognizable by designing a TM algorithm.

Decidable Languages

We have seen many examples of decidable languages:

- Languages about strings
- Languages about DFAs/NFAs/PDAs/CFGs – know which ones are decidable and which are not, why
- Be comfortable with TM's that take another machine as input

Relationships Among Language Classes



Outline

Preliminaries – Countable and Uncountable Sets

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

Preliminaries – 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: $\mathcal{N} = 1, 2, 3, \ldots$

Preliminaries – 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

Preliminaries – 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*

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

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...
⋮	⋮

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 $x_i \neq f(i)_i$

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 $\mathcal{N}$

Take Aways

L_{TM} is Turing-recognizable

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

L_{TM} is Turing-recognizable

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

Proof: By construction of machine $M_{L_{TM}}$

$M_{L_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w

L_{TM} is Turing-recognizable

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

Proof: By construction of machine $M_{L_{TM}}$

$M_{L_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

L_{TM} is Turing-recognizable

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

Proof: By construction of machine $M_{L_{TM}}$

$M_{L_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in L_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.

L_{TM} is Turing-recognizable

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

Proof: By construction of machine $M_{L_{TM}}$

$M_{L_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in L_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.
- Hence, $M_{L_{TM}}$, also halts and outputs 1

L_{TM} is Turing-recognizable

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

Proof: By construction of machine $M_{L_{TM}}$

$M_{L_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in L_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.
- Hence, $M_{L_{TM}}$, also halts and outputs 1
- Thus, $M_{L_{TM}}$ accepts all inputs in L_{TM}

L_{TM} is Turing-recognizable

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

Proof: By construction of machine $M_{L_{TM}}$

$M_{L_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in L_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.
- Hence, $M_{L_{TM}}$, also halts and outputs 1
- Thus, $M_{L_{TM}}$ accepts all inputs in L_{TM}
- Note that $M_{L_{TM}}$ may not halt on all inputs – doesn't decide L_{TM}

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

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

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 $\langle D\rangle$

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

Outline

Main Observation

Suppose that $A \leq B$, then:

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

- Suppose that B is decidable

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

- Suppose that B is decidable
- Since $A \leq B$, there exists an algorithm (i.e., a reduction) that uses a solution to B to solve A

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

- Suppose that B is decidable
- Since $A \leq B$, there exists an algorithm (i.e., a reduction) that uses a solution to B to solve A
- But, this means that A is decidable by running the machine for B as needed by the reduction

Outline

1-Slide Summary

- DFA=NFA=Regular Languages = Regular expressions
- PDA = CFG

1-Slide Summary

- DFA=NFA=Regular Languages = Regular expressions
- PDA = CFG

For the exam

- Remember what each of the terms above means
- Remember how they work
- Remember relationships between them
- There will be no pumping lemma questions on the exam

1-Slide Summary

- DFA=NFA=Regular Languages = Regular expressions
- PDA = CFG

For the exam

- Remember what each of the terms above means
- Remember how they work
- Remember relationships between them
- There will be no pumping lemma questions on the exam

Exam

Exam Details:

- Tuesday, May 9, 10:20-12:20
- In the classroom
- 2 sheets (back-and-front) of notes are allowed

See you all there!