BBM402-Lecture 11: The Class NP

Lecturer: Lale Özkahya

Resources for the presentation:
http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/Syllabus/
https://courses.engr.illinois.edu/cs498374/lectures.html

Introduction

- P = { L | there is some polynomial-time deterministic Turing machine that decides L }
- NP = { L | there is some polynomial-time nondeterministic Turing machine that decides L }
- Alternatively, $L \in NP$ if and only if ($\exists$ V, a polynomial-time verifier) ($\exists$ p, a polynomial) such that:

  $x \in L$ iff $(\exists c, |c| \leq p(|x|))$ [V(x, c) accepts]

  certificate

- To show that $L \in NP$, we need only exhibit a suitable verifier V and show that it works (which requires saying what the certificates are).
- $P \subseteq NP$, but it's not known whether P = NP.

Introduction

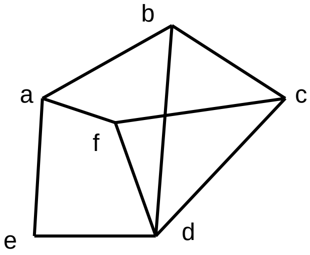
- P = { L | $\exists$ poly-time deterministic TM that decides L }
- NP = { L | $\exists$ poly-time nondeterministic TM that decides L }
- L $\in$ NP if and only if ($\exists$ V, poly-time verifier) ($\exists$ p, poly)

  $x \in L$ iff ($\exists$ c, $|c| \leq p(|x|)$) [V(x, c) accepts]

- Some languages are in NP, but are not known to be in P (and are not known to not be in P):
  - SAT = { $< \phi >$ | ϕ is a satisfiable Boolean formula }
  - 3COLOR = { $< G >$ | G is an (undirected) graph whose vertices can be colored with $\leq$ 3 colors with no 2 adjacent vertices colored the same }
  - CLIQUE = { $< G, k >$ | G is a graph with a k-clique }
  - VERTEX-COVER = { $< G, k >$ | G is a graph having a vertex cover of size k }

CLIQUE

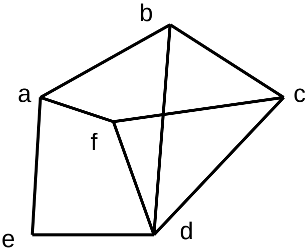
- CLIQUE = { < G, k > | G is a graph with a k-clique }
- k-clique: k vertices with edges between all pairs in the clique.
- In NP, not known to be in P, not known to not be in P.



- 3-cliques: { b, c, d }, { c, d, f }
- Cliques are easy to verify, but may be hard to find.

CLIQUE

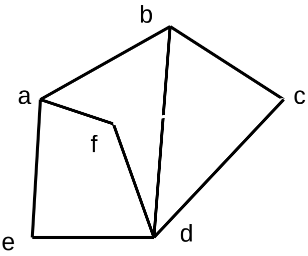
- CLIQUE = { < G, k > | G is a graph with a k-clique }



- Input to the VC problem: < G, 3 >
- Certificate, to show that < G, 3 > $\in$ CLIQUE, is { b, c, d } (or { c, d, f }).
- Polynomial-time verifier can check that { b, c, d } is a 3-clique.

VERTEX-COVER

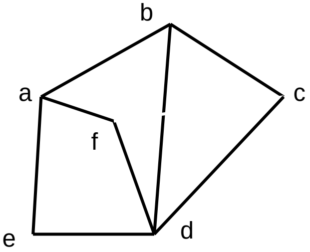
- VERTEX-COVER = { < G, k > | G is a graph with a vertex cover of size k }
- Vertex cover of G = (V, E): A subset C of V such that, for every edge (u,v) in E, either u $\in$ C or v $\in$ C.
  - A set of vertices that "covers" all the edges.
- In NP, not known to be in P, not known to not be in P.



- 3-vc: { a, b, d }
- Vertex covers are easy to verify, may be hard to find.

VERTEX-COVER

- VERTEX-COVER = { < G, k > | G is a graph with a vertex cover of size k }



- Input to the VC problem: < G, 3 >
- Certificate, to show that < G, 3 > $\in$ VC, is { a, b, d }.
- Polynomial-time verifier can check that { a, b, d } is a 3-vertex-cover.

HAMPATH Problem

HAMPATH $=$
$\{< G, s, t > | G$is a directed path with a Hamiltonian $s, t -$ path$\}$

- There is an exponential brute force algorithm given in Thm 7.14 (Sipser).
- **Verifying the existence of a Hamiltonian path is much easier than determining its existence.**

HAMPATH Problem

HAMPATH $=$
$\{< G, s, t > | G$is a directed path with a Hamiltonian $s, t -$ path$\}$

- There is an exponential brute force algorithm given in Thm 7.14 (Sipser).
- **Verifying the existence of a Hamiltonian path is much easier than determining its existence.**

Verifier: A verifier for a language A is an algorithm, where

$$A = \{w | V \text{ accepts } < w, c > \text{ for some string c}\}$$

HAMPATH Problem

HAMPATH $=$
$\{< G, s, t > | G$is a directed path with a Hamiltonian $s, t -$ path$\}$

- There is an exponential brute force algorithm given in Thm 7.14 (Sipser).
- **Verifying the existence of a Hamiltonian path is much easier than determining its existence.**

Verifier: A verifier for a language A is an algorithm, where

$$A = \{w | V \text{ accepts } < w, c > \text{ for some string c}\}$$

Polynomial time verifier: The running time of the verifier is measured by the length of w. Thus, a poly. time verifier runs in $O(w^k)$ (polynomial time in the length of w).

Certificate: A verifier uses additional information, represented by the symbol c. This information is called a *certificate*, or proof, of membership in A.

Why is HAMPATH in NP?

- NP is the class of languages that have polynomial time verifiers.
- The term NP comes from "nondeterministic polynomial time".

Decider for HAMPATH

N : *"On input* $< G, s, t >$*, where* G *is a directed graph with nodes* s *and* t*:*

1. *Write a list of* m *numbers* $p_1, \ldots, p_m$*, where* m *is the number of nodes in* G*. Each number in the list is nondeterministically selected between 1 and* m*.*
2. *Check for repetitions in the list. If any are found, reject.*
3. *Check whether* $s = p_1$ *and* $t = p_m$*. If either fail, reject.*
4. *For each* $1 \leq i \leq m - 1$*, check whether* (p_i, p_{i+1}) *is an edge of* G*. If they are not, reject. Otherwise, all tests have been passed, so accept.*

Why is HAMPATH in NP?

- NP is the class of languages that have polynomial time verifiers.
- The term NP comes from "nondeterministic polynomial time".

Decider for HAMPATH

N : "On input $< G, s, t >$, where G is a directed graph with nodes s and t:

1. *Write a list of m numbers $p_1, \ldots, p_m$, where m is the number of nodes in G. Each number in the list is nondeterministically selected between 1 and m.*
2. *Check for repetitions in the list. If any are found, reject.*
3. *Check whether $s = p_1$ and $t = p_m$. If either fail, reject.*
4. *For each $1 \leq i \leq m - 1$, check whether (p_i, p_{i+1}) is an edge of G. If they are not, reject. Otherwise, all tests have been passed, so accept.*

This algorithm runs in nondeterministic polynomial time:
- Stage 1: The nondeterministic selection runs in p.t.
- Stage 2, 3 and 4: p.t.

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\implies \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\implies$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\implies \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\implies$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

- Let V be the poly. time verifier for A that exists because $A \in NP$, assuming V runs in time $O(n^k)$.

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\implies \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\implies$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

- Let V be the poly. time verifier for A that exists because $A \in NP$, assuming V runs in time $O(n^k)$.
- N = "On input w of length n: (1) Nondeterministically select string c of length at most n^k (depth of the tree modelling the nondeterministic machine) (2) Run V on input $< w, c >$. (3) If V accepts, *accept*; otherwise, *reject*.

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\Longrightarrow \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\Longrightarrow$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

- Let V be the poly. time verifier for A that exists because $A \in NP$, assuming V runs in time $O(n^k)$.
- N = "On input w of length n: (1) Nondeterministically select string c of length at most n^k (depth of the tree modelling the nondeterministic machine) (2) Run V on input $< w, c >$. (3) If V accepts, *accept*; otherwise, *reject*.

To show (2): Assume that A is decided by a poly. time NTM N. Construct a polynomial time verifier V for A.

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\implies \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\implies$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

- Let V be the poly. time verifier for A that exists because $A \in NP$, assuming V runs in time $O(n^k)$.
- N = "On input w of length n: (1) Nondeterministically select string c of length at most n^k (depth of the tree modelling the nondeterministic machine) (2) Run V on input $< w, c >$. (3) If V accepts, *accept*; otherwise, *reject*.

To show (2): Assume that A is decided by a poly. time NTM N. Construct a polynomial time verifier V for A.
On input $< w, c >$, where w and c are strings:

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\Longrightarrow \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\Longrightarrow$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

- Let V be the poly. time verifier for A that exists because $A \in NP$, assuming V runs in time $O(n^k)$.
- N = "On input w of length n: (1) Nondeterministically select string c of length at most n^k (depth of the tree modelling the nondeterministic machine) (2) Run V on input $< w, c >$. (3) If V accepts, *accept*; otherwise, *reject*.

To show (2): Assume that A is decided by a poly. time NTM N. Construct a polynomial time verifier V for A.
On input $< w, c >$, where w and c are strings:

- Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step,

More about NP

Theorem (Thm 7.20, Sipser)

A language is in NP "iff" (if and only if) it is decided by some nondeterministic polynomial time machine. ((1) language in NP $\implies \exists$ nondet. p. t. machine AND (2) Nondet. p.t. machine $\implies$ language in NP)

To show (1): Let $A \in NP$. Show that A is decided by a poly. time NTM N.

- Let V be the poly. time verifier for A that exists because $A \in NP$, assuming V runs in time $O(n^k)$.
- N = "On input w of length n: (1) Nondeterministically select string c of length at most n^k (depth of the tree modelling the nondeterministic machine) (2) Run V on input $< w, c >$. (3) If V accepts, *accept*; otherwise, *reject*.

To show (2): Assume that A is decided by a poly. time NTM N. Construct a polynomial time verifier V for A.

On input $< w, c >$, where w and c are strings:

- Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step,
- If this branch of N's computation accepts, *accept*; otherwise, *reject*.

Some Classical Optimization Problems

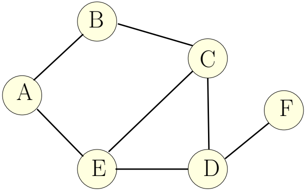
- Maximum Independent Set
- Maximum Clique
- Minimum Vertex Cover
- Traveling Salesman Problem
- Knapsack Problems
- Integer Linear Programming
- ...

All of these optimization problems have a decision version which is an **NP** problem. And there are many, many other problems too.

Maximum Independent Set in a Graph

Definition

Given undirected graph $\mathbf{G = (V, E)}$ a subset of nodes $\mathbf{S} \subseteq \mathbf{V}$ is an independent set (also called a stable set) if for there are no edges between nodes in $\mathbf{S}$. That is, if $\mathbf{u, v} \in \mathbf{S}$ then $\mathbf{(u, v)} \notin \mathbf{E}$.

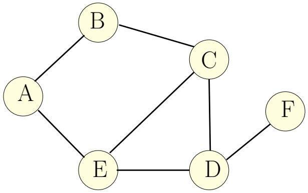


Some independent sets in graph above: $\mathbf{\{D\}, \{A, C\}, \{B, E, F\}}$

Maximum Independent Set Problem

Input Graph $\mathbf{G = (V, E)}$

Goal Find maximum sized independent set in $\mathbf{G}$



MIS is an optimization problem. How do we cast it as a decision problem?

Decision version of Maximum Independent Set

Input Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer $\mathbf{k}$ written as $\mathbf{G}, \mathbf{k}$

Question Is there an independent set in $\mathbf{G}$ of size at least $\mathbf{k}$?

The answer to $< \mathbf{G}, \mathbf{k} >$ is YES if $\mathbf{G}$ has an independent set of size at least $\mathbf{k}$. Otherwise the answer is NO. Sometimes we say $< \mathbf{G}, \mathbf{k} >$ is a YES instance or a NO instance.

The language associated with this decision problem is

$$\mathbf{L}_{\mathbf{MIS}} = \{< \mathbf{G}, \mathbf{k} >| \ \mathbf{G} \text{ has an independent set of size} \geq \mathbf{k}\}$$

MIS is in NP

$\mathbf{L_{MIS}} = \{< \mathbf{G}, \mathbf{k} >| \mathbf{G}$ has an independent set of size $\geq \mathbf{k}\}$
A non-deterministic polynomial-time algorithm for $\mathbf{L_{MIS}}$.

Input: a string $< \mathbf{G}, \mathbf{k} >$ encoding graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer $\mathbf{k}$

1. Non-deterministically guess a subset $\mathbf{S} \subseteq \mathbf{V}$ of vertices
2. Verify (deterministically) that
    1. $\mathbf{S}$ forms an independent set in $\mathbf{G}$ by checking that there is no edge in $\mathbf{E}$ between any pair of vertices in $\mathbf{S}$
    2. $|\mathbf{S}| \geq \mathbf{k}$.
3. If $\mathbf{S}$ passes the above two tests output YES Else NO

MIS is in NP

$\mathbf{L_{MIS}} = \{< \mathbf{G}, \mathbf{k} >| \mathbf{G}$ has an independent set of size $\geq \mathbf{k}\}$
A non-deterministic polynomial-time algorithm for $\mathbf{L_{MIS}}$.

Input: a string $< \mathbf{G}, \mathbf{k} >$ encoding graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer $\mathbf{k}$

1. Non-deterministically guess a subset $\mathbf{S} \subseteq \mathbf{V}$ of vertices
2. Verify (deterministically) that
   1. $\mathbf{S}$ forms an independent set in $\mathbf{G}$ by checking that there is no edge in $\mathbf{E}$ between any pair of vertices in $\mathbf{S}$
   2. $|\mathbf{S}| \geq \mathbf{k}$.
3. If $\mathbf{S}$ passes the above two tests output YES Else NO

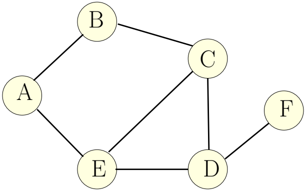
Key points:

- string encoding $\mathbf{S}$, $< \mathbf{S} >$ has length polynomial in length of input $< \mathbf{G}, \mathbf{k} >$
- verification of guess is easily seen to be polynomial in length of $< \mathbf{S} >$ and $< \mathbf{G}, \mathbf{k} >$.

Minimum Vertex Cover

Definition

Given undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ a subset of nodes $\mathbf{S} \subseteq \mathbf{V}$ is an vertex cover if every edge $(\mathbf{u}, \mathbf{v})$ has at least one of its end points in $\mathbf{S}$. That is, every edge is covered by $\mathbf{S}$.

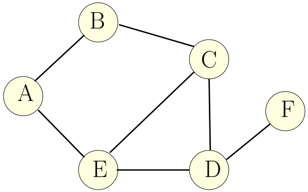


Examples of vertex covers in graph above:

Minimum Vertex Cover

Input Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Goal Find minimum sized vertex cover in $\mathbf{G}$



Decision version: given $\mathbf{G}$ and $\mathbf{k}$, does $\mathbf{G}$ have a vertex cover of size *at most* $\mathbf{k}$?

$$\mathbf{L_{VC}} = \{< \mathbf{G}, \mathbf{k} >| \ \mathbf{G} \text{ has a vertex cover size} \leq \mathbf{k}\}$$

Minimum Vertex Cover is in NP

$\mathbf{L_{VC}} = \{< \mathbf{G}, \mathbf{k} > | \ \mathbf{G}$ has a vertex cover size $\leq \mathbf{k}\}$
A non-deterministic polynomial-time algorithm for $\mathbf{L_{VC}}$.

Input: a string $< \mathbf{G}, \mathbf{k} >$ encoding graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer $\mathbf{k}$

1. Non-deterministically guess a subset $\mathbf{S} \subseteq \mathbf{V}$ of vertices
2. Verify (deterministically) that
    1. $\mathbf{S}$ forms a vertex cover in $\mathbf{G}$ by checking that for each edge $(\mathbf{u}, \mathbf{v}) \in \mathbf{E}$ at least one of $\mathbf{u}, \mathbf{v}$ is in $\mathbf{S}$
    2. $|\mathbf{S}| \leq \mathbf{k}$.
3. If $\mathbf{S}$ passes the above two tests output YES Else NO

Minimum Vertex Cover is in NP

$\mathbf{L_{VC}} = \{ < \mathbf{G}, \mathbf{k} > | \ \mathbf{G}$ has a vertex cover size $\leq \mathbf{k}\}$
A non-deterministic polynomial-time algorithm for $\mathbf{L_{VC}}$.

Input: a string $< \mathbf{G}, \mathbf{k} >$ encoding graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer $\mathbf{k}$

1. Non-deterministically guess a subset $\mathbf{S} \subseteq \mathbf{V}$ of vertices
2. Verify (deterministically) that
   1. $\mathbf{S}$ forms a vertex cover in $\mathbf{G}$ by checking that for each edge $(\mathbf{u}, \mathbf{v}) \in \mathbf{E}$ at least one of $\mathbf{u}, \mathbf{v}$ is in $\mathbf{S}$
   2. $|\mathbf{S}| \leq \mathbf{k}$.
3. If $\mathbf{S}$ passes the above two tests output YES Else NO

Key points:

- string encoding $\mathbf{S}$, $< \mathbf{S} >$ has length polynomial in length of input $< \mathbf{G}, \mathbf{k} >$
- verification of guess is easily seen to be polynomial in length of $< \mathbf{S} >$ and $< \mathbf{G}, \mathbf{k} >$.

Sudoku

			2	5				
	3	6		4		8		
	4					1	6	
2								
7	6						1	9
								3
	1	5					7	
		9		8		2	4	
				3	7			

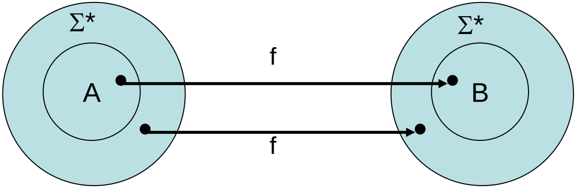
Given $\mathbf{n} \times \mathbf{n}$ sudoku puzzle, does it have a solution?

Polynomial-Time Reducibility

Polynomial-Time Reducibility

- Definition: $A \subseteq \Sigma^*$ is polynomial-time reducible to $B \subseteq \Sigma^*$, $A \leq_p B$, provided there is a polynomial-time computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that:

  $(\forall w)$ [$w \in A$ if and only if $f(x) \in B$]



- Extends to different alphabets Σ_1 and Σ_2.
- Same as mapping reducibility, $\leq_m$, but with a polynomial-time restriction.

Polynomial-Time Reducibility

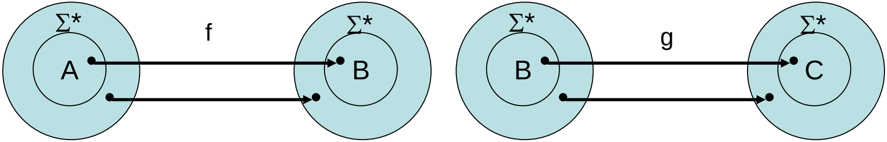
- Definition: $A \subseteq \Sigma^*$ is polynomial-time reducible to $B \subseteq \Sigma^*$, $A \leq_p B$, provided there is a polynomial-time computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that:

  $$(\forall w)\ [\ w \in A \text{ if and only if } f(x) \in B\ ]$$

- Theorem: (Transitivity of $\leq_p$)

  If $A \leq_p B$ and $B \leq_p C$ then $A \leq_p C$.

- Proof:
  - Let f be a polynomial-time reducibility function from A to B.
  - Let g be a polynomial-time reducibility function from B to C.

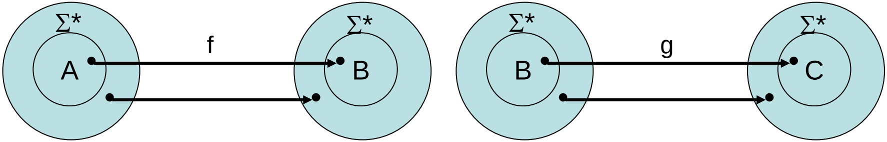


Polynomial-Time Reducibility

- Definition: $A \leq_p B$, provided there is a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$ such that:

  $$(\forall w)\ [\ w \in A \text{ if and only if } f(w) \in B\ ]$$

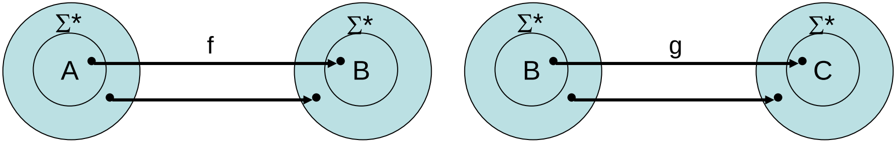
- Theorem: If $A \leq_p B$ and $B \leq_p C$ then $A \leq_p C$.
- Proof:
  - Let f be a polynomial-time reducibility function from A to B.
  - Let g be a polynomial-time reducibility function from B to C.



  - Define $h(w) = g(f(w))$.
  - Then $w \in A$ if and only if $f(w) \in B$ if and only if $\underbrace{g(f(w))}_{\boxed{h(w)}} \in C$.
  - h is poly-time computable:

Polynomial-Time Reducibility

- Theorem: If $A \leq_p B$ and $B \leq_p C$ then $A \leq_p C$.
- Proof:
  - Let f be a polynomial-time reducibility function from A to B.
  - Let g be a polynomial-time reducibility function from B to C.



  - Define h(w) = g(f(w)).
  - h is poly-time computable:
    - |f(w)| is bounded by a polynomial in |w|.
    - Time to compute g(f(w)) is bounded by a polynomial in |f(w)|, and therefore by a polynomial in |w|.
    - Uses the fact that substituting one polynomial for the variable in another yields yet another polynomial.

Polynomial-Time Reducibility

- Definition: $A \leq_p B$, provided there is a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$ such that:

  $$(\forall w) \, [\, w \in A \text{ if and only if } f(x) \in B \,]$$

- Theorem: If $A \leq_p B$ and $B \in P$ then $A \in P$.
- Proof:
  - Let f be a polynomial-time reducibility function from A to B.
  - Let M be a polynomial-time decider for B.
  - To decide whether $w \in A$:
    - Compute $x = f(w)$.
    - Run M to decide whether $x \in B$, and accept / reject accordingly.
  - Polynomial time.

- Corollary: If $A \leq_p B$ and A is not in P then B is not in P.
- Easiness propagates downward, hardness propagates upward.

Polynomial-Time Reducibility

- Can use $\leq_p$ to relate the difficulty of two problems:
- Theorem: If $A \leq_p B$ and $B \leq_p A$ then either both A and B are in P or neither is.
- Also, for problems in NP:
- Theorem: If $A \leq_p B$ and $B \in NP$ then $A \in NP$.
- Proof:
  - Let f be a polynomial-time reducibility function from A to B.
  - Let M be a polynomial-time nondeterministic TM that decides B.
    - Poly-bounded on all branches.
    - Accepts on at least one branch iff and only if input string is in B.
  - NTM M′ to decide membership in A:
  - On input w:
    - Compute $x = f(w)$; $|x|$ is bounded by a polynomial in $|w|$.
    - Run M on x and accept/reject (on each branch) if M does.
  - Polynomial time-bounded NTM.

Polynomial-Time Reducibility

- Theorem: If $A \leq_p B$ and $B \in NP$ then $A \in NP$.
- Proof:
  - Let f be a polynomial-time reducibility function from A to B.
  - Let M be a polynomial-time nondeterministic TM that decides B.
  - NTM M′ to decide membership in A:
  - On input w:
    - Compute x = f(w); |x| is bounded by a polynomial in |w|.
    - Run M on x and accept/reject (on each branch) if M does.
  - Polynomial time-bounded NTM.
  - Decides membership in A:
    - M′ has an accepting branch on input w
      iff M has an accepting branch on f(w), by definition of M′,
      iff $f(w) \in B$, since M decides B,
      iff $w \in A$, since $A \leq_p B$ using f.
  - So M′ is a poly-time NTM that decides A, $A \in NP$.

Polynomial-Time Reducibility

- Theorem: If $A \leq_p B$ and $B \in NP$ then $A \in NP$.
- Corollary: If $A \leq_p B$ and A is not in NP, then B is not in NP.

Polynomial-Time Reducibility

- A technical result (curiosity):
- Theorem: If $A \in P$ and B is any nontrivial language (meaning not $\emptyset$, not Σ^*), then $A \leq_p B$.
- Proof:
  - Suppose $A \in P$.
  - Suppose B is a nontrivial language; pick $b_0 \in B$, $b_1 \in B^c$.
  - Define $f(w) = b_0$ if $w \in A$, b_1 if w is not in A.
  - f is polynomial-time computable; why?
  - Because A is polynomial time decidable.
  - Clearly $w \in A$ if and only if $f(w) \in B$.
  - So $A \leq_p B$.
- Trivial reduction: All the work is done by the decider for A, not by the reducibility and the decider for B.