Chapter 24

coNP, Self-Reductions

CS 473: Fundamental Algorithms, Spring 2013
April 24, 2013

24.1 Complementation and Self-Reduction

24.2 Complementation

24.2.1 Recap

24.2.1.1 The class P

(A) A language L (equivalently decision problem) is in the class **P** if there is a polynomial time algorithm A for deciding L; that is given a string x, A correctly decides if $x \in L$ and running time of A on x is polynomial in $|x|$, the length of x.

24.2.1.2 The class NP

Two equivalent definitions:

(A) Language L is in **NP** if there is a non-deterministic polynomial time algorithm A (Turing Machine) that decides L.
  (A) For $x \in L$, A has some non-deterministic choice of moves that will make A accept x
  (B) For $x \notin L$, no choice of moves will make A accept x

(B) L has an efficient certifier $C(\cdot, \cdot)$.
  (A) C is a polynomial time deterministic algorithm
  (B) For $x \in L$ there is a string y (proof) of length polynomial in $|x|$ such that $C(x, y)$ accepts
  (C) For $x \notin L$, no string y will make $C(x, y)$ accept

24.2.1.3 Complementation

Definition 24.2.1. *Given a decision problem X, its* **complement** $\overline{X}$ *is the collection of all instances s such that $s \notin L(X)$*

Equivalently, in terms of languages:

Definition 24.2.2. *Given a language L over alphabet Σ, its* **complement** $\overline{L}$ *is the language $\Sigma^* \setminus L$.*

24.2.1.4 Examples

(A) $\textsf{PRIME} = \{n \mid n \text{ is an integer and } n \text{ is prime}\}$
$\overline{\textsf{PRIME}} = \left\{n \,\middle|\, n \text{ is an integer and } n \text{ is not a prime}\right\}$
$\overline{\textsf{PRIME}} = \textsf{COMPOSITE}$.

(B) $\textsf{SAT} = \left\{\varphi \,\middle|\, \varphi \text{ is a CNF formula and } \varphi \text{ is satisfiable}\right\}$
$\overline{\textsf{SAT}} = \left\{\varphi \,\middle|\, \varphi \text{ is a CNF formula and } \varphi \text{ is not satisfiable}\right\}$.
$\overline{\textsf{SAT}} = \textsf{UnSAT}$.

Technicality: $\overline{\textsf{SAT}}$ also includes strings that do not encode any valid CNF formula. Typically we ignore those strings because they are not interesting. In all problems of interest, we assume that it is "easy" to check whether a given string is a valid instance or not.

24.2.1.5 P is closed under complementation

Proposition 24.2.3. *Decision problem X is in* **P** *if and only if $\overline{X}$ is in* **P**.

Proof:
(A) If X is in **P** let A be a polynomial time algorithm for X.
(B) Construct polynomial time algorithm A' for $\overline{X}$ as follows: given input x, A' runs A on x and if A accepts x, A' rejects x and if A rejects x then A' accepts x.
(C) Only if direction is essentially the same argument.

■

24.2.2 Motivation

24.2.2.1 Asymmetry of NP

Definition 24.2.4. ***Nondeterministic Polynomial Time*** *(denoted by* **NP***) is the class of all problems that have efficient certifiers.*

Observation To show that a problem is in **NP** we only need short, efficiently checkable certificates for "yes"-instances. What about "no"-instances?

Given a CNF formula φ, is φ unsatisfiable?

Easy to give a proof that φ is satisfiable (an assignment) but no easy (known) proof to show that φ is unsatisfiable!

24.2.2.2 Examples of complement problems

Some languages

(A) **UnSAT**: CNF formulas φ that are not satisfiable
(B) **No-Hamilton-Cycle**: graphs G that do not have a Hamilton cycle
(C) **No-3-Color**: graphs G that are not 3-colorable

Above problems are complements of known **NP** problems (viewed as languages).

24.2.3 co-NPDefinition

24.2.3.1 NP and co-NP

NP Decision problems with a polynomial certifier.

Examples: **SAT**, **Hamiltonian Cycle**, **3-Colorability**.

Definition 24.2.5. **co-NP** *is the class of all decision problems* X *such that* $\overline{X} \in$ **NP**.

Examples: **UnSAT**, **No-Hamiltonian-Cycle**, **No-3-Colorable**.

24.2.4 Relationship between P, NP and co-NP

24.2.4.1 co-NP

If L is a language in **co-NP** then that there is a polynomial time certifier/verifier $C(\cdot,\cdot)$, such that:

(A) for $s \notin L$ there is a proof t of size polynomial in $|s|$ such that $C(s,t)$ correctly says NO.
(B) for $s \in L$ there is no proof t for which $C(s,t)$ will say NO

co-NP has checkable proofs for strings NOT in the language.

24.2.4.2 Natural Problems in co-NP

(A) **Tautology**: given a Boolean formula (not necessarily in CNF form), is it true for *all* possible assignments to the variables?
(B) **Graph expansion**: given a graph G, is it an *expander*? A graph $G = (V, E)$ is an ***expander*** if and only if for each $S \subset V$ with $|S| \leq |V|/2$, $|N(S)| \geq |S|$. Expanders are very important graphs in theoretical computer science and mathematics.

24.2.4.3 Factorization, Primality

Problem: Primality

Instance: An integer n. **Question:** Is the number n prime?

Problem: Factoring

Instance: Integers n, k. **Question:** Does the number n has a factor $\leq k$? Formally, is there ℓ, such that $2 \leq \ell \leq k$, such that ℓ divides n?

(A) **Primality** is in **P**.
(B) **Factoring** is in **NP** $\cap$ **co-NP**.

24.2.4.4 Factoring is a very naughty problem

Problem: Factoring

> **Instance:** Integers n, k.
> **Question:** Does the number n has a factor $\leq k$? Formally, is there ℓ, such that $2 \leq \ell \leq k$, such that ℓ divides n?

If answer is:

(A) NO: certificate is all prime factors of n. Certification: multiply the given numbers.
(B) YES: Certificate is the factor ℓ. Verify it divides n.

Belief: Unlikely **Factoring** is **NP-Complete**. Can be solved in polynomial time on a quantum computer.

24.2.4.5 P, NP, co-NP

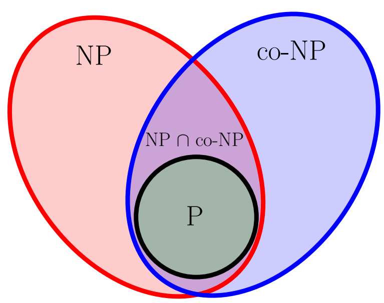
co-P: complement of P. Language X is in **co-P** iff $\overline{X} \in P$

Proposition 24.2.6. **P** = **co-P**.

Proposition 24.2.7. **P** $\subseteq$ **NP** $\cap$ **co-NP**.

Saw that **P** $\subseteq$ **NP**. Same proof shows **P** $\subseteq$ **co-NP**.

24.2.4.6 P, NP, and co-NP



Open Problems:

(A) Does **NP** = **co-NP**?
Consensus opinion: No.
(B) Is **P** = **NP** $\cap$ **co-NP**?
No real consensus.

24.2.4.7 P, NP, and co-NP

Proposition 24.2.8. *If* **P** = **NP** *then* **NP** = **co-NP**.

Proof: **P** = **co-P**
If **P** = **NP** then **co-NP** = **co-P** = **P**. ∎

24.2.5 P, NP, and co-NP

24.2.5.1 Which means that...

Corollary 24.2.9. *If* **NP** $\neq$ **co-NP** *then* **P** $\neq$ **NP**.

Importance of corollary: try to prove **P** $\neq$ **NP** by proving that **NP** $\neq$ **co-NP**.

24.2.5.2 NP $\cap$ co-NP

Complexity Class NP $\cap$ co-NP Problems in this class have
(A) Efficient certifiers for yes-instances
(B) Efficient disqualifiers for no-instances
Problems have a **good characterization** property, since for both yes and no instances we have short efficiently checkable proofs.

24.2.5.3 NP $\cap$ co-NP: Example

Example 24.2.10. **Bipartite Matching**: *Given bipartite graph* $G = (U \cup V, E)$*, does* G *have a perfect matching?*

Bipartite Matching $\in$ **NP** $\cap$ **co-NP**

(A) If G *is a yes-instance, then proof is just the perfect matching.*
(B) If G *is a no-instance, then by Hall's Theorem, there is a subset of vertices* $A \subseteq U$ *such that* $|N(A)| < |A|$.

Example 24.2.11 (More interesting...). **Factoring** $\in$ **NP**$\cap$**co-NP**, *and we do not know if it is in* **P**!

24.2.5.4 Good Characterization $\stackrel{?}{=}$ Efficient Solution

(A) Bipartite Matching has a polynomial time algorithm
(B) Do all problems in **NP** $\cap$ **co-NP** have polynomial time algorithms? That is, is **P** $=$ **NP** $\cap$ **co-NP**?
Problems in **NP** $\cap$ **co-NP** have been proved to be in **P** many years later
  (A) Linear programming (Khachiyan 1979)
    (A) Duality easily shows that it is in **NP** $\cap$ **co-NP**
  (B) Primality Testing (Agarwal-Kayal-Saxena 2002)
    (A) Easy to see that **PRIME** is in **co-NP** (why?)
    (B) **PRIME** is in **NP** - not easy to show! (Vaughan Pratt 1975)

24.2.5.5 P $\stackrel{?}{=}$ NP $\cap$ co-NP (contd)

(A) Some problems in **NP** $\cap$ **co-NP** still cannot be proved to have polynomial time algorithms
  (A) Parity Games.
  (B) Other more specialized problems.

24.2.5.6 co-NP Completeness

Definition 24.2.12. *A problem X is said to be* ***co-NP-Complete*** *(**co-NPC**) if*
(A) $X \in$ **co-NP**
(B) ***(Hardness)*** *For any $Y \in$ **co-NP**, $Y \leq_P X$*

co-NP-Complete problems are the hardest problems in **co-NP**.

Lemma 24.2.13. *X is **co-NPC** if and only if $\overline{X}$ is **NP-Complete**.*

Proof left as an exercise.

24.2.5.7 P, NP and co-NP

Possible scenarios:
(A) **P** = **NP**. Then **P** = **NP** = **co-NP**.
(B) **NP** = **co-NP** and **P** ≠ **NP** (and hence also **P** ≠ **co-NP**).
(C) **NP** ≠ **co-NP**. Then **P** ≠ **NP** and also **P** ≠ **co-NP**.
Most people believe that the last scenario is the likely one.
Question: Suppose **P** ≠ **NP**. Is every problem that is in **NP** \ **P** is also **NP-Complete**?

Theorem 24.2.14 (Ladner). *If **P** ≠ **NP** then there is a problem/language $X \in$ **NP** \ **P** such that X is not **NP-Complete**.*

24.2.5.8 Karp vs Turing Reduction and NP vs co-NP

Question: Why restrict to Karp reductions for NP-Completeness?

Lemma 24.2.15. *If $X \in$ **co-NP** and Y is **NP-Complete** then $X \leq_P Y$ under Turing reduction.*

Thus, Turing reductions cannot distinguish **NP** and **co-NP**.

24.3 Self Reduction

24.3.1 Introduction

24.3.1.1 Back to Decision versus Search

(A) Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance

Example 24.3.1. *(A) Satisfiability*
 (A) ***Decision:*** *Is the formula φ satisfiable?*
 (B) ***Search:*** *Find assignment that satisfies φ*
(B) Graph coloring
 (A) ***Decision:*** *Is graph G 3-colorable?*
 (B) ***Search:*** *Find a 3-coloring of the vertices of G*

24.3.1.2 Decision "reduces to" Search

(A) Efficient algorithm for search implies efficient algorithm for decision.
(B) If decision problem is difficult then search problem is also difficult.
(C) Can an efficient algorithm for decision imply an efficient algorithm for search?
Yes, for all the problems we have seen. In fact for all **NP-Complete** Problems.

24.3.2 Self Reduction

24.3.2.1 Self Reduction

Definition 24.3.2. *A problem is said to be* **self reducible** *if the search problem reduces (by Turing reduction) in polynomial time to decision problem. In other words, there is an algorithm to solve the search problem that has polynomially many steps, where each step is either*
(A) A conventional computational step, or
(B) a call to subroutine solving the decision problem.

24.3.3 SAT is Self Reducible

24.3.3.1 Back to SAT

Proposition 24.3.3. **SAT** *is self reducible.*

In other words, there is a polynomial time algorithm to find the satisfying assignment if one can periodically check if some formula is satisfiable.

24.3.4 Search Algorithm for SAT

24.3.4.1 given a Decision Algorithm for SAT

Input: **SAT** formula φ with n variables $x_1, x_2, \ldots, x_n$.
(A) set $x_1 = 0$ in φ and get new formula φ_1. check if φ_1 is satisfiable using decision algorithm. if φ_1 is satisfiable, recursively find assignment to $x_2, x_3, \ldots, x_n$ that satisfy φ_1 and output $x_1 = 0$ along with the assignment to $x_2, \ldots, x_n$.
(B) if φ_1 is not satisfiable then set $x_1 = 1$ in φ to get formula φ_2. if φ_2 is satisfiable , recursively find assignment to $x_2, x_3, \ldots, x_n$ that satisfy φ_2 and output $x_1 = 1$ along with the assignment to $x_2, \ldots, x_n$.
(C) if φ_1 and φ_2 are both not satisfiable then φ is not satisfiable.

Algorithm runs in polynomial time if the decision algorithm for **SAT** runs in polynomial time. At most $2n$ calls to decision algorithm.

24.3.4.2 Self-Reduction for NP-Complete Problems

Theorem 24.3.4. *Every* **NP-Complete** *problem/language L is self-reducible.*

Proof is not hard but requires understanding of proof of Cook-Levin theorem.

Note that proof is only for complete languages, not for all languages in **NP**. Otherwise **Factoring** would be in polynomial time and we would not rely on it for our current security protocols.

Easy and instructive to prove self-reducibility for specific **NP-Complete** problems such as **Independent Set**, **Vertex Cover**, **Hamiltonian Cycle**, etc.

See discussion section problems.