- Since the class NP is not known to be closed under complementation, motivating us to define and stury the following complexity class: coNP $= \{L : \overline{L} \in \text{NP}\}$.

* $\overline{\text{SAT}} = \{< \phi > \mid \phi$ is a boolean formula in conjunctive normal form and ϕ is not satisfiable$\}$, $\overline{\text{HAMPATH}}$, $\overline{\text{SUBSETSUM}} \in$ coNP.

  Since $\overline{\text{TAUTOLOGY}} = \{< \phi >:$ there is a truth assignment for which the boolean formula ϕ is false$\}$ belongs to NP,
  TAUTOLOGY = $\{< \phi >:$ boolean formula ϕ is satisfiable by every truth assignment$\} \in$ coNP.

* $L \in$ NP: for every $w \in L$, there is a proof (certificate) c of length polynomial in $|w|$ so that a polynomial time verifier (a DTM) for L on input $< w, c >$ could verify w belongs to L

  $L \in$ coNP: for every $w \notin L$, there is a proof (certificate) c of length polynomial in $|w|$ so that a polynomial time verifier (a DTM) for $\overline{L}$ on input $< w, c >$ could verify w belongs to $\overline{L}$ (that is, every no instance w of L has a short refutation to claim $w \notin L$)

  $L \in \text{NP} \cap \text{coNP}$: both the yes and no instances w of L have proofs (certificates) of length polynomial in $|w|$

* Most believe NP $\neq$ coNP since it does not seem that a short certificate could exist to verify a given boolean formula belongs to TAUTOLOGY.
  (To remind, most believe P $\subsetneq$ NP due to following reasons: (i) verifying solution to a problem is in general easier to finding a solution itself; (ii) P is closed under complement whereas NP is not known to be closed under complementation: given *any* polynomial time NTM based decider N (that is, $L(N)$ in NP), constructing a polynomial time NTM based decider for the complement of $L(N)$ is open.
  Noting a branch of computation rejecting input would let other branches continue computing, and a branch accepting halts all branches, toggling accept states to reject states and reject states to accept states in an NTM not necessarily decide $\overline{L(N)}$. Significantly, rejecting along a branch doesn't necessarily mean NTM will reject the input and hence it is not right to accept along that branch. That is, NTM model of computation does not have a provision to accept the input whenever *all* branches reject.)

- P $\subseteq$ NP $\cap$ coNP.

  proof: L $\in$ P $\Rightarrow$ $\overline{\text{L}} \in$ P (since P is closed under complement) $\Rightarrow$ $\overline{\text{L}} \in$ NP $\Rightarrow$ L $\in$ coNP

- If P $=$ NP then NP $=$ coNP. That is, if NP $\neq$ coNP then P $\neq$ NP.

  proof:

  NP $\subseteq$ coNP: L $\in$ NP $\Rightarrow$ L $\in$ P (since P=NP) $\Rightarrow$ $\overline{\text{L}} \in$ P $\Rightarrow$ $\overline{\text{L}} \in$ NP $\Rightarrow$ L $\in$ coNP

  coNP $\subseteq$ NP: L $\in$ coNP $\Rightarrow$ $\overline{\text{L}} \in$ NP $\Rightarrow$ $\overline{\text{L}} \in$ P (since P=NP) $\Rightarrow$ $L \in$ P $\Rightarrow$ L $\in$ NP

- contrapositive: If NP $\neq$ coNP then P $\neq$ NP.

- coNP $\subseteq$ PSPACE.

  proof: NP $\subseteq$ PSPACE $\Rightarrow$ coNP $\subseteq$ coPSPACE; however, coPSPACE $=$ PSPACE

- $\overline{\text{PRIMES}}$ = COMPOSITES = $\{< n > \mid n$ is a positive integer and n is a composite number$\} \in$ NP

* PRIMES = $\{< n > \mid n$ is a positive integer and n is a prime number$\} \in$ NP $\leftarrow$ for Pratt's algorithm, refer to [HU] or the last page of this note

- Till '02, PRIMES problem was only known to be in NP $\cap$ coNP (but not known to be in class P). This is when AKS[1] algorithm proved PRIMES is in indeed in class P.[2]

- INTFACTORIZATION = $\{< n, a, b > \mid n, a, b$ are positive integers encoded in binary and there is a prime number $p \in [a, b]$ that divides $n\} \in$ NP

* $\overline{\text{INTFACTORIZATION}}$ = $\{< n, a, b > \mid n, a, b$ are positive integers encoded in binary and there is no prime number $p \in [a, b]$ that divides $n\} \in$ NP

- As of now, there is no known polynomial time algorithm for the integer factorization problem, hence this problem is known to be in NP $\cap$ coNP but we do not know whether it belongs to class P. The best factoring algorithm runs in time $2^{O((\lg n)^{1/3}\sqrt{\lg \lg n})}$.

- A language L is said to be coNP-hard if $\forall_{L' \in coNP} L' \leq_p L$, equivalently, $\forall_{\overline{L'} \in NP} \overline{L'} \leq_p \overline{L}$. However, this says, $\overline{L}$ is NP-hard.

  Therefore, L is coNP-hard iff $\overline{L}$ is NP-hard. Hence, to prove L is coNP-hard, one could prove $\overline{L}$ is NP-hard.

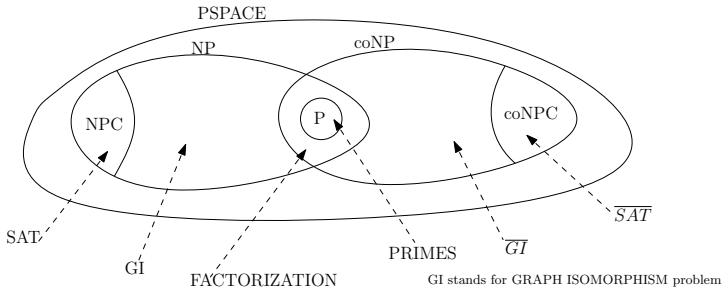
* A language L is said to be coNP-complete whenever $L \in$ coNP and L is coNP-hard.

* $L \in$ NP-complete $\Leftrightarrow \overline{L} \in$ coNP-complete.

* TAUTOLOGY is coNP-complete.

  proof: we already proved TAUTOLOGY $\in$ coNP; since SAT is NP-hard, we give $SAT \leq_p \overline{\text{TAUTOLOGY}}$, equivalently, $\overline{\text{SAT}} \leq_p$ TAUTOLOGY; $\phi \in \overline{\text{SAT}}$ iff $f(\phi) = \neg\phi \in$ TAUTOLOGY, for mapping reduction f that can be computed in polynomial time

- In conclusion, this is what the general belief or the conjectured relation of these classes is:



[1] named after M. Agarwal, N. Kayal, and N. Saxena

[2] Narendra Karmarkar's weakly polynomial time algorithm for linear programming, found in '84, is another result by our countrymen, which is also as famous as this one is.

Again, we do not have any proofs to claim whether NP $\neq$ coNP, P $\neq$ NP $\cap$ coNP, P $\neq$ NP, P $\neq$ coNP, NP $\subset$ PSPACE, coNP $\subset$ PSPACE, ...

- An NP-complete language is in coNP iff NP = coNP.

  $\Leftarrow$ obvious

  $\Rightarrow$ proof of NP $\subseteq$ coNP: let L be the language that is NP, NP-hard, and coNP; since L is NP-hard, $\forall_{L' \in NP} L' \leq_p L$, equivalently, $\forall_{L' \in NP} \overline{L'} \leq_p \overline{L}$; since $L \in coNP$, $\overline{L} \in NP$; hence, $\overline{L}$ has an NTM based polynomial time decider; a polynomial time NTM for $\overline{L'}$ on input x would first compute $f(x)$ (here f is a polynomial time reduction function) and supply it to the NTM for $\overline{L}$, therefore, $\overline{L'} \in$ NP

  $\Rightarrow$ proof of coNP $\subseteq$ NP: – homework –

* Corollary: Consistent with the general belief of NP $\neq$ coNP, till date, for no NP-complete language L, the $\overline{L}$ is known to be in NP.

Pratt's algorithm

Given a positive integer n in binary, the nondeterministic polynomial time algorithm devised herewith determines whether n is a prime.

- Fermat's little theorem: $n > 2$ is a prime iff for some integer $x \in (1, n)$,

  $x^{n-1} \equiv 1 \mod n$, and ——— (1)

  $x^i \not\equiv 1 \mod n$, for all i, $1 \leq i < n-1$. ———- (2)

- however, checking (2) explicitly is computationally inefficient

- hence, noting below proposition

  $x^{(n-1)/p_j} \not\equiv 1 \mod n$ for every prime p_j in the prime decomposition of $n-1$ ———- (3)[3]

  implies (2), in designing algorithm, Pratt verifies (3) instead of (2)

- Observations: the input size is $\lg n$ since n is in binary; in any prime decomposition of $n-1$, there are at most $\lg(n-1)$ numbers (since each number need to be at least 2) ; and, the size of every such factor is at most $\lg(n-1)$

- isPrime(n): [n is encoded in binary]

  (i) if $n = 2$ then return true [denoting n is a prime]

  (ii) if $n = 1$ or n is an even integer > 2 then return false

  (iii) nondeterministically guess an $x \in (1, n)$

  (iv) if $x^{n-1} \equiv 1 \mod n$ [due to (1)]

  (a) nondeterministically guess at most $k' = \lg(n-1)$ numbers, each $\in [2, n-1)$; let $n_1, \ldots, n_{k'}$ be these numbers; reject if $n_1 \cdot n_2 \cdot \ldots \cdot n_{k'} \neq n-1$

[3] none of (2), (3), and (3) $\Rightarrow$ (2) were proved in lectures

(b) for any $j \in [1, k']$ if isPrime(n_j) is false then reject

//reaches here if $n_1 n_2 \ldots n_{k'}$ is a prime decompositon of $n-1$

(c) if $x^{(n-1)/n_j} \not\equiv 1 \mod n$ for every $j \in [1, k']$, then return true [due to (3)]

(v) return false

- time analysis: since modular exponentiation takes $O((\lg n)^3)$ time, excluding recursive calls, this code takes $O((\lg n)^3)$ time; let $t(\ell)$ be the time along any branch of nondeterministic computation for input of size ℓ; then,

$$t(\lg n) = O((\lg n)^3) + t(\lg n_1) + t(\lg n_2) + \ldots + t(\lg n_{k'})$$

$$= (\lg n)^3 + (\lg n_1)^4 + (\lg n_2)^4 + \ldots + (\lg n_{k'})^4 \quad \text{[guessing } t(\lg r) \text{ is } (\lg r)^4 \text{ for any positive } r\text{, and substituting]}$$

$$\leq O((\lg n)^3) + (\lg (n_1 n_2 \ldots n_{k'}))^4$$

$$= O((\lg n)^3) + (\lg (n-1))^4$$

$$= O((\lg n)^4)$$