A CS 310 presentation

Complexity Hierarchy

Soham Joshi

10.1.1 Problems Solvable in Polynomial Time

A Turing machine M is said to be of *time complexity* $T(n)$ [or to have "running time $T(n)$"] if whenever M is given an input w of length n, M halts after making at most $T(n)$ moves, regardless of whether or not M accepts. This definition applies to any function $T(n)$, such as $T(n) = 50n^2$ or $T(n) = 3^n + 5n^4$; we shall be interested predominantly in the case where $T(n)$ is a polynomial in n. We say a language L is in class $\mathcal{P}$ if there is some polynomial $T(n)$ such that $L = L(M)$ for some deterministic TM M of time complexity $T(n)$.

10.1.3 Nondeterministic Polynomial Time

A fundamental class of problems in the study of intractability is those problems that can be solved by a nondeterministic TM that runs in polynomial time. Formally, we say a language L is in the class $\mathcal{NP}$ (nondeterministic polynomial) if there is a nondeterministic TM M and a polynomial time complexity $T(n)$ such that $L = L(M)$, and when M is given an input of length n, there are no sequences of more than $T(n)$ moves of M.

- Every det TM = non-det TM

$\Rightarrow$ $P \subseteq NP$

TSP

The input to TSP is the same as to MWST, a graph with integer weights on the edges such as that of Fig. 10.1, and a weight limit W. The question asked is whether the graph has a "Hamilton circuit" of total weight at most W. A *Hamilton circuit* is a set of edges that connect the nodes into a single cycle,

On the other hand, if we had a nondeterministic computer, we could guess a permutation of the nodes, and compute the total weight for the cycle of nodes in that order. If there were a real computer that was nondeterministic, no branch would use more than $O(n)$ steps if the input was of length n. On a multitape NTM, we can guess a permutation in $O(n^2)$ steps and check its total weight in a similar amount of time. Thus, a single-tape NTM can solve the TSP in $O(n^4)$ time at most. We conclude that the TSP is in $\mathcal{NP}$.

Polytime reductions

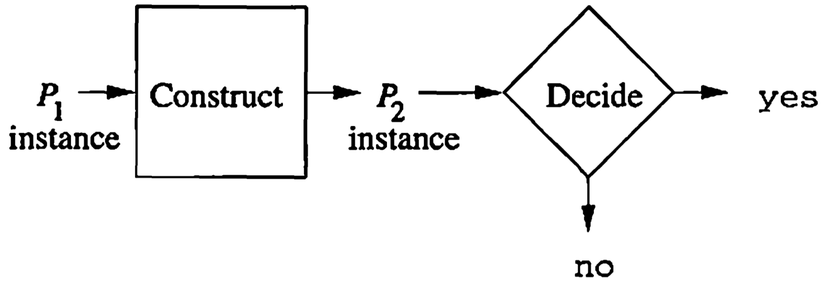

Figure 10.2: Reprise of the picture of a reduction

Caveat: if instance of size $m \to 2^m$, then $P_2 \in P$, $P_1 \notin P$ is possible if time for construction $\notin$ poly (input size)

The correct restriction to place on the translation from P_1 to P_2 is that it requires time that is polynomial in the length of its input. Note that if the translation takes time $O(m^j)$ on input of length m, then the output instance of P_2 cannot be longer than the number of steps taken, i.e., it is at most cm^j for some constant c. Now, we can prove that if P_2 is in $\mathcal{P}$, then so is P_1.

NP Completeness

L is *NP-complete* if the following statements are true about L:

1. L is in $\mathcal{NP}$.
2. For every language L' in $\mathcal{NP}$ there is a polynomial-time reduction of L' to L.

Theorem 10.4: If P_1 is NP-complete, P_2 is in $\mathcal{NP}$, and there is a polynomial-time reduction of P_1 to P_2, then P_2 is NP-complete.

Proof. Follows by composition of polynomial t^n is a polynomial.

Theorem 10.5: If some NP-complete problem P is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$.

PROOF: Suppose P is both NP-complete and in $\mathcal{P}$. Then all languages L in $\mathcal{NP}$ reduce in polynomial-time to P. If P is in $\mathcal{P}$, then L is in $\mathcal{P}$, as we discussed in Section 10.1.5. □

SAT is in NP

The *boolean expressions* are built from:

1. Variables whose values are boolean; i.e., they either have the value 1 (true) or 0 (false).
2. Binary operators $\wedge$ and $\vee$, standing for the logical AND and OR of two expressions.
3. Unary operator $\neg$ standing for logical negation.
4. Parentheses to group operators and operands, if necessary to alter the default precedence of operators: $\neg$ highest, then $\wedge$, and finally $\vee$.

The *satisfiability problem* is:

- Given a boolean expression, is it satisfiable?

1. The symbols $\wedge$, $\vee$, $\neg$, (, and) are represented by themselves.
2. The variable x_i is represented by the symbol x followed by 0's and 1's that represent i in binary.

} encoding

Theorem 10.9: (Cook's Theorem) SAT is NP-complete.

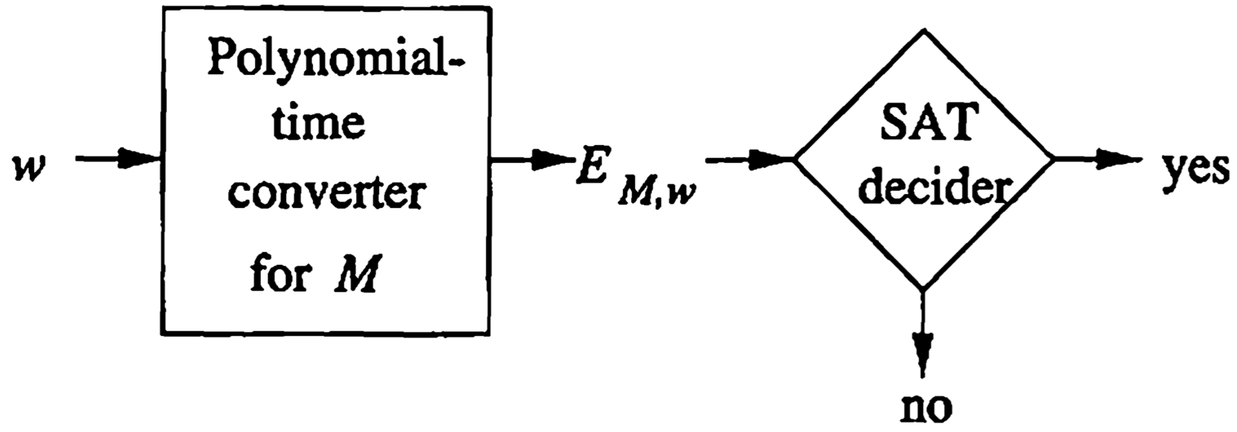

Figure 10.5: If SAT is in $\mathcal{P}$, then every language in $\mathcal{NP}$ could be shown to be in $\mathcal{P}$ by a DTM designed in this manner

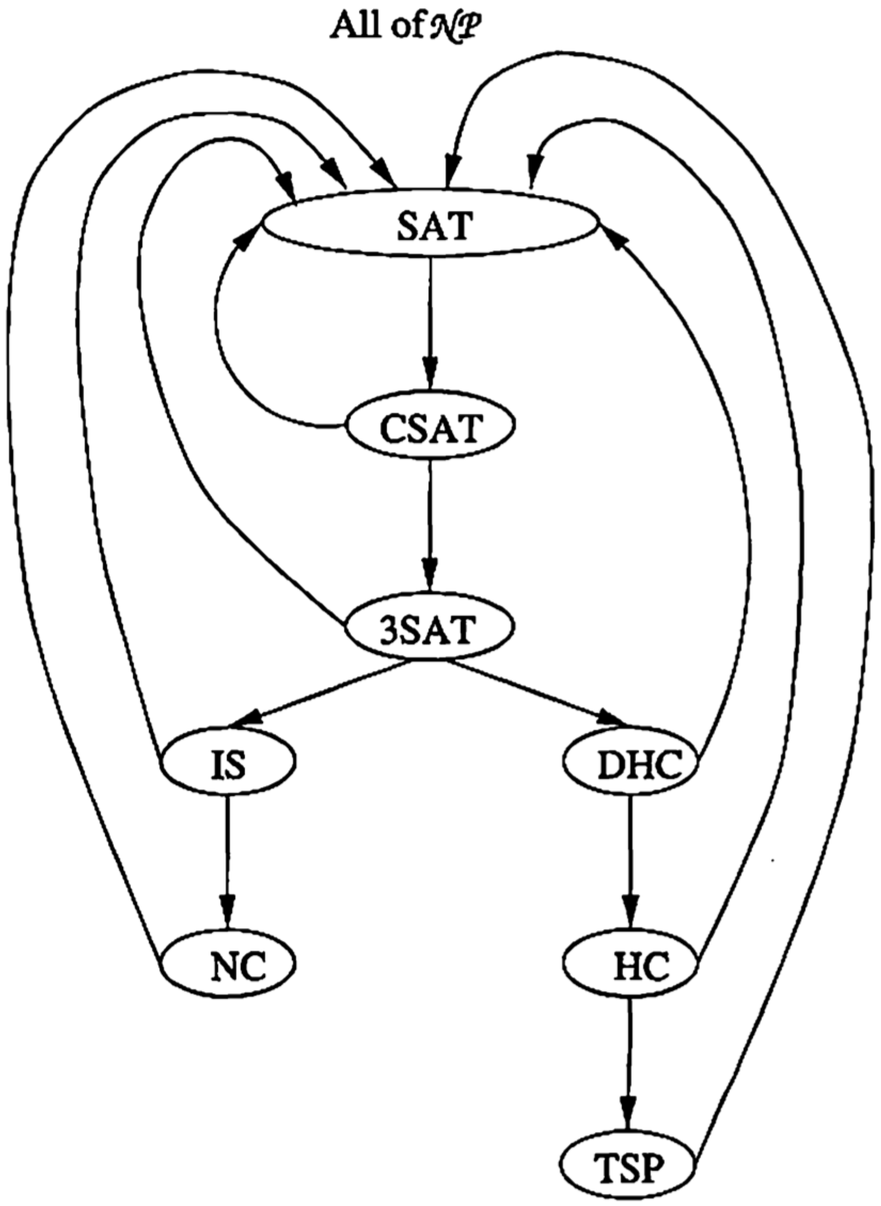

Figure 10.12: Reductions among NP-complete problems

Beyond Classes P and NP

Co-NP

- Languages *whose* complement is NP
- Even if P is not equal to NP, NP and co-NP could still be equal, but we haven't found even one NP complete problem whose complement is in Co-NP

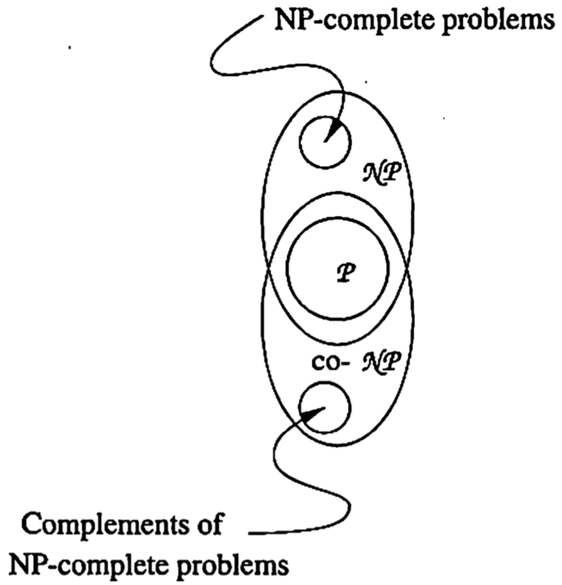

Figure 11.1: Suspected relationship between co-$\mathcal{NP}$ and other classes of languages

Theorem 11.2: $\mathcal{NP} = \text{co-}\mathcal{NP}$ if and only if there is some NP-complete problem whose complement is in $\mathcal{NP}$.

(⇒) SAT ∈ NP

(⇐) Use PTIME reductions

PSPACE (PS) and NPS

A polynomial-space-bounded Turing machine is suggested by Fig. 11.2. There is some polynomial $p(n)$ such that when given input w of length n, the TM never visits more than $p(n)$ cells of its tape. By Theorem 8.12, we may assume that the tape is semi-infinite, and the TM never moves left from the beginning of its input.

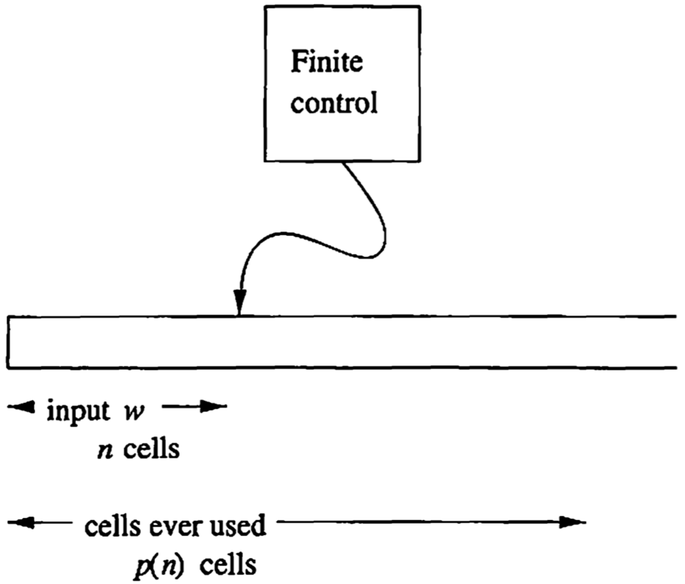

Figure 11.2: A TM that uses polynomial space

Define the class of languages $\mathcal{PS}$ (*polynomial space*) to include all and only the languages that are $L(M)$ for some polynomial-space-bounded, deterministic Turing machine M. Also, define the class $\mathcal{NPS}$ (*nondeterministic polynomial space*) to consist of those languages that are $L(M)$ for some nondeterministic, polynomial-space-bounded TM M. Evidently $\mathcal{PS} \subseteq \mathcal{NPS}$, since every deterministic TM is technically nondeterministic also. However, we shall prove the surprising result that $\mathcal{PS} = \mathcal{NPS}$.[1]

Adding these folks to our hierarchy

To start, the relationships $\mathcal{P} \subseteq \mathcal{PS}$ and $\mathcal{NP} \subseteq \mathcal{NPS}$ should be obvious. The reason is that if a TM makes only a polynomial number of moves, then it uses no more than a polynomial number of cells; in particular, it cannot visit more cells than one plus the number of moves it makes. Once we prove $\mathcal{PS} = \mathcal{NPS}$, we shall see that in fact the three classes form a chain of containment: $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PS}$.

Theorem 11.3: If M is a polynomial-space-bounded TM (deterministic or nondeterministic), and $p(n)$ is its polynomial space bound, then there is a constant c such that if M accepts its input w of length n, it does so within $c^{1+p(n)}$ moves.

Theorem 11.4: If L is a language in $\mathcal{PS}$ (respectively $\mathcal{NPS}$), then L is accepted by a polynomial-space-bounded deterministic (respectively nondeterministic) TM that halts after making at most $c^{q(n)}$ moves, for some polynomial $q(n)$ and constant $c > 1$.

Theorem 11.5: (*Savitch's Theorem*) $\mathcal{PS} = \mathcal{NPS}$.

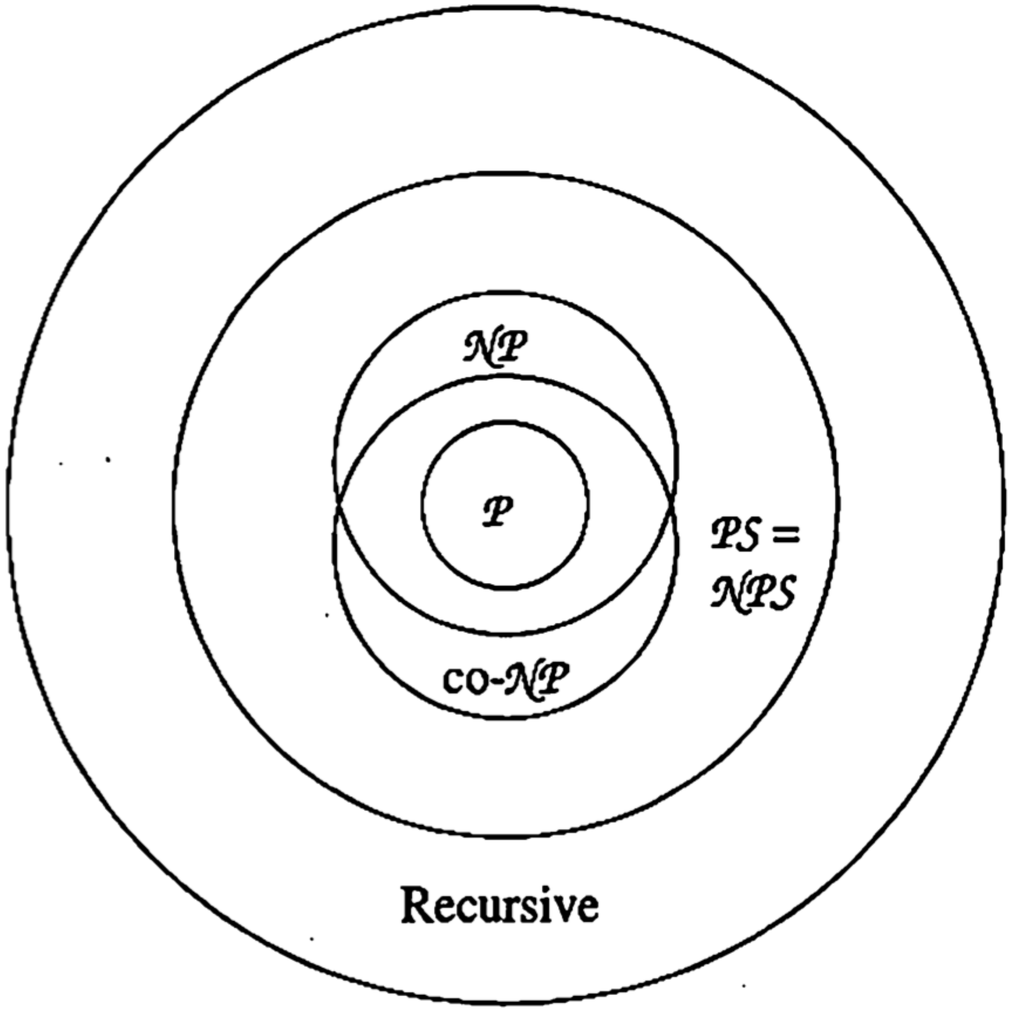

Figure 11.5: Known relationships among classes of languages

PS Completeness

We define a problem P to be *complete for* $\mathcal{PS}$ (PS-complete) if:

1. P is in $\mathcal{PS}$.
2. All languages L in $\mathcal{PS}$ are polynomial-time reducible to P.

(why not polynomial sp. reduction?)

Theorem 11.6: Suppose P is a PS-complete problem. Then:

a) If P is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{PS}$.

b) If P is in $\mathcal{NP}$, then $\mathcal{NP} = \mathcal{PS}$.

← Motivation for having polytime in PS completeness

Example of PS Complete (QBF)

QBF : $\forall$, $\exists$ quantified boolean formulas

QBF problem : Is the value of the formula, 1 ?

Theorem 11.10: QBF is in $\mathcal{PS}$.

Theorem 11.11: The problem QBF is PS-complete.

Adding Randomisation (RP and ZPP)

Random bits have prior ∞ work or internal coin flips every time head moves

For a word w, it is accepted with probability p_w.

Non-deterministic acceptance is similar to randomized acceptance with $p > 0$.

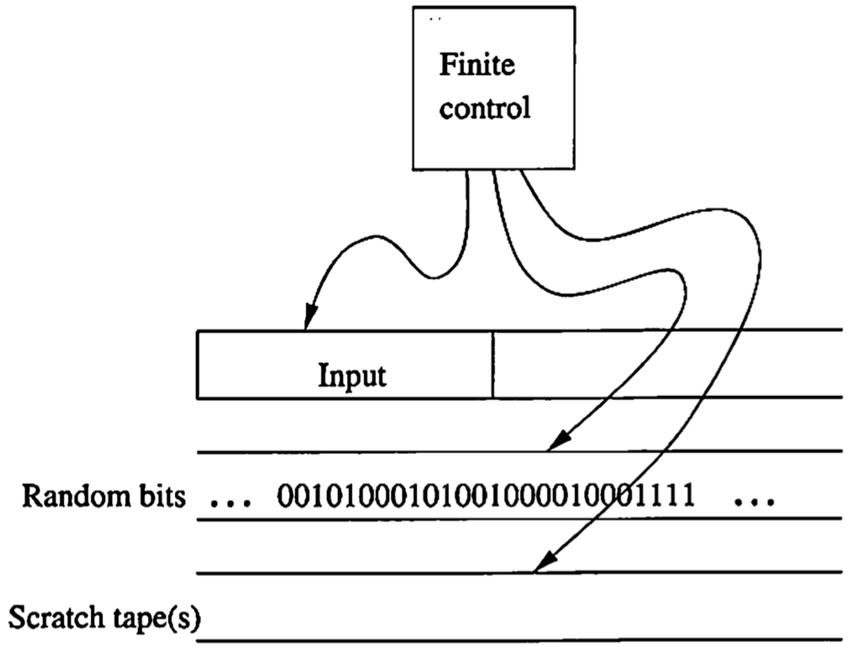

Figure 11.6: A Turing machine with the capability of using randomly "generated" numbers

RP

Acceptance Rules

1. If w is not in L, then the probability that M accepts w is 0.
2. If w is in L, then the probability that M accepts w is at least $1/2$.
3. There is a polynomial $T(n)$ such that if input w is of length n, then all runs of M, regardless of the contents of the random tape, halt after at most $T(n)$ steps.

of running time, we may say that a randomized TM is "Monte-Carlo" if it either accepts with probability 0 or accepts with probability at least 1/2, with nothing in between. Point (3) simply addresses the running time, which is independent of whether or not the TM is "Monte-Carlo."

Membership for RP

1. If w is not in L, then our run will surely not lead to acceptance of w.
2. If w is in L, there is at least a 50% chance that w will be accepted.

Theorem 11.16: If L is in $\mathcal{RP}$, then for any constant $c > 0$, no matter how small, there is a polynomial-time randomized algorithm that renders a decision whether its given input w is in L, makes no false-positive errors, and makes false-negative errors with probability no greater than c. □

Pf : (Repeated testing)

ZPP (zero error, probabilistic, polynomial)

always halts, and has an expected time to halt that is some polynomial in the length of the input. This TM accepts its input if it enters and accepting state (and therefore halts at that time), and it rejects its input if it halts without accepting. Thus, the definition of class $\mathcal{ZPP}$ is almost the same as the definition of $\mathcal{P}$, except that $\mathcal{ZPP}$ allows the behavior of the TM to involve randomness, and the expected running time, rather than the worst-case running time is measured.

If $L \in ZPP$, $\overline{L} \in ZPP$

Theorem 11.17: $\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}$.

(Relating ZP to RP, co-RP)

Relation to P and NP

Theorem 11.17 tells us that $\mathcal{ZPP} \subseteq \mathcal{RP}$. We can place these classes between $\mathcal{P}$ and $\mathcal{NP}$ by the following simple theorems.

Theorem 11.18: $\mathcal{P} \subseteq \mathcal{ZPP}$.

PROOF: Any deterministic, polynomial-time bounded TM is also a Las-Vegas, polynomial-time bounded TM, that happens not to use its ability to make random choices. □

Theorem 11.19: $\mathcal{RP} \subseteq \mathcal{NP}$.

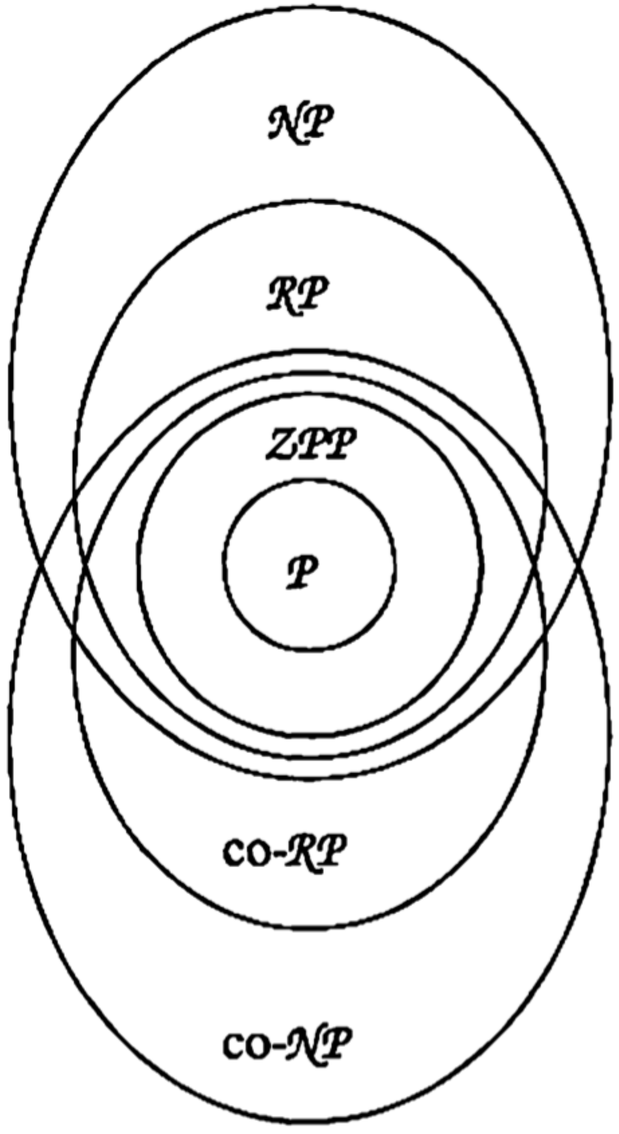

Figure 11.8: Relationship of $\mathcal{ZPP}$ and $\mathcal{RP}$ to other classes