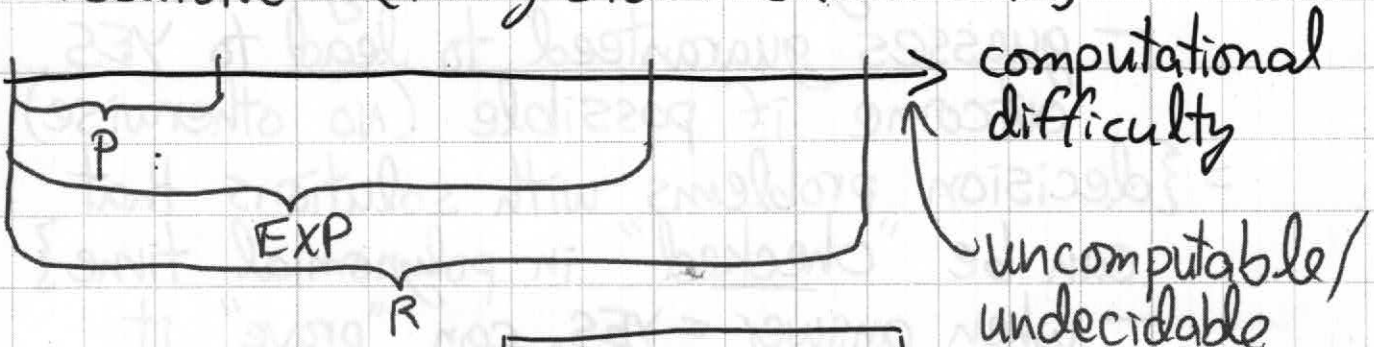
Intro to complexity:

$P = \{$problems solvable in polynomial time$\}$ ($\to n^c$)

$EXP = \{$problems solvable in exponential time$\}$ ($\to 2^{n^c}$)

$R = \{$problems solvable in finite time$\}$

↳ "recursive" [Turing 1936; Church 1941]



$$P \subsetneq EXP \subsetneq R$$

Examples:

- negative-weight cycle detection $\in P$
- n×n Chess $\in EXP$ but $\notin P$
  - ↳ who wins from given board config.?
- Tetris $\in EXP$ but don't know whether $\in P$
  - ↳ survive given pieces from given board
- halting problem* $\notin R$
- "most" decision problems $\notin R$
  (# algorithms $\approx \mathbb{N}$; # dec. problems $\approx 2^{\mathbb{N}} = \mathbb{R}$)

*Halting problem: the problem of determining from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever. (All descriptions are on Turing Machine). The problem is undecidable over Turing machine.

P = {decision problems solvable in poly time}

All problems with answer ∈ {YES, NO}

NP = {decision problems solvable in poly. time via a "lucky" algorithm} ← Also called a certificate

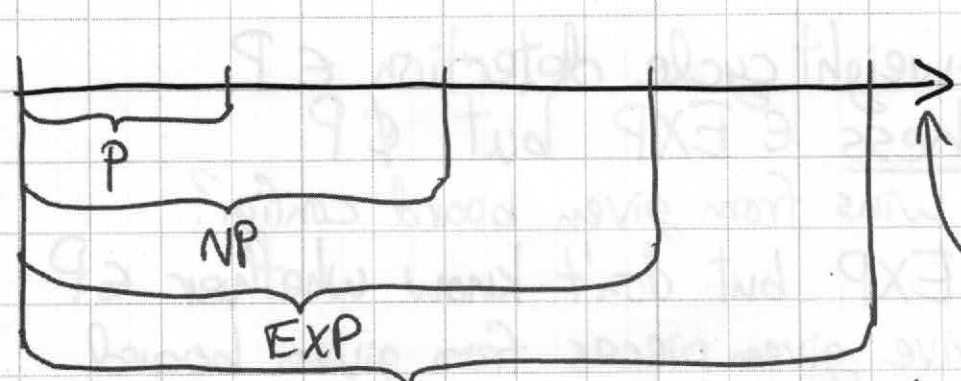
NOT non-polynomial

↳ can make lucky guesses, always "right", without trying all options ← often a solution

- nondeterministic model: algorithm makes guesses & then says YES or NO
- guesses guaranteed to lead to YES outcome if possible (NO otherwise)

= {decision problems with solutions that can be "checked" in polynomial time}

- when answer = YES, can "prove" it & poly.-time algorithm can check proof



In Tetris, the sequence of pieces are given and a board of size $n \times m$ ($m \leq n$) and an initial configuration and the question is of surviving? (any completely filled row is cleared and all pieces above drops by one row.

Example: Tetris ∈ NP

- nondeterministic alg: - guess each move
  - did I survive?
- proof of YES: list what moves to make (rules of Tetris are easy)

(by Dynamic Prog.)

In typical Tetris, $n \times m = 20 \times 10$, but for constant n the problem is in P

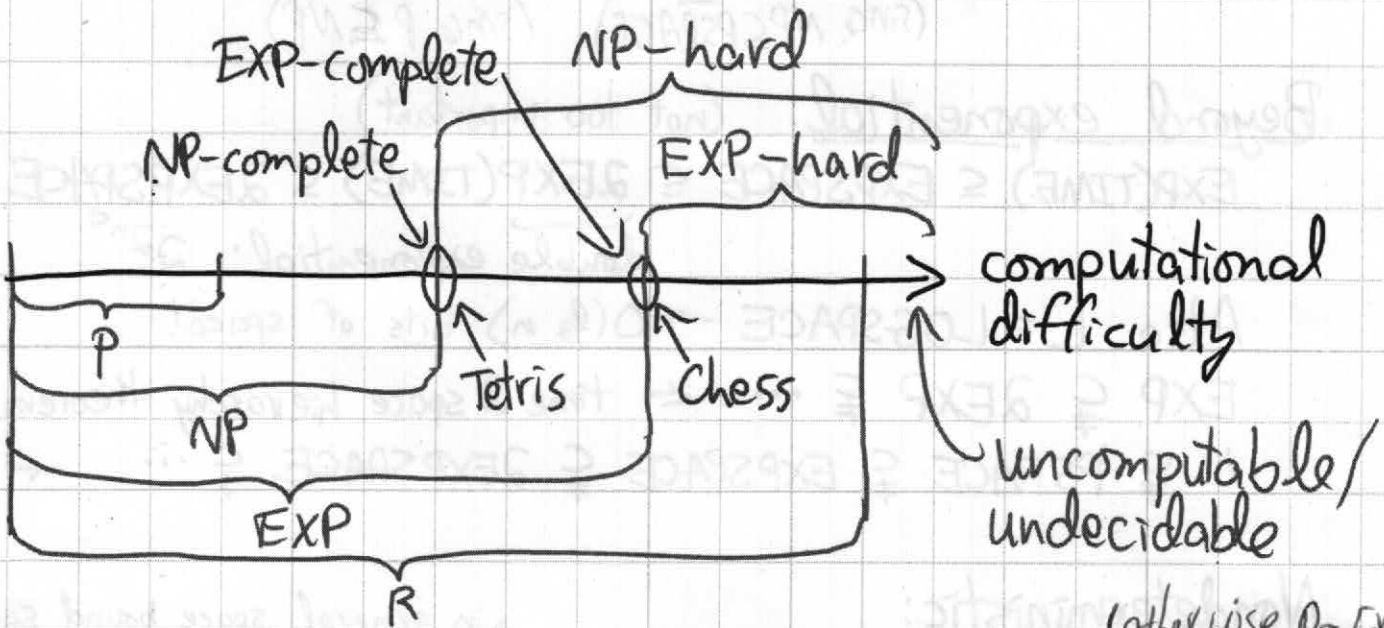
So far everything in NP ∩ CoNP ∈ P

P ≠ NP: big conjecture (worth $1,000,000)
≈ can't engineer luck
≈ generating (proofs of) solutions can be harder than checking them

CoNP = negations (YES ↔ NO) of problems ∈ NP
= problems with good proofs of No answer
(e.g. Is there a unique solution for 3-SAT or Coloring?)

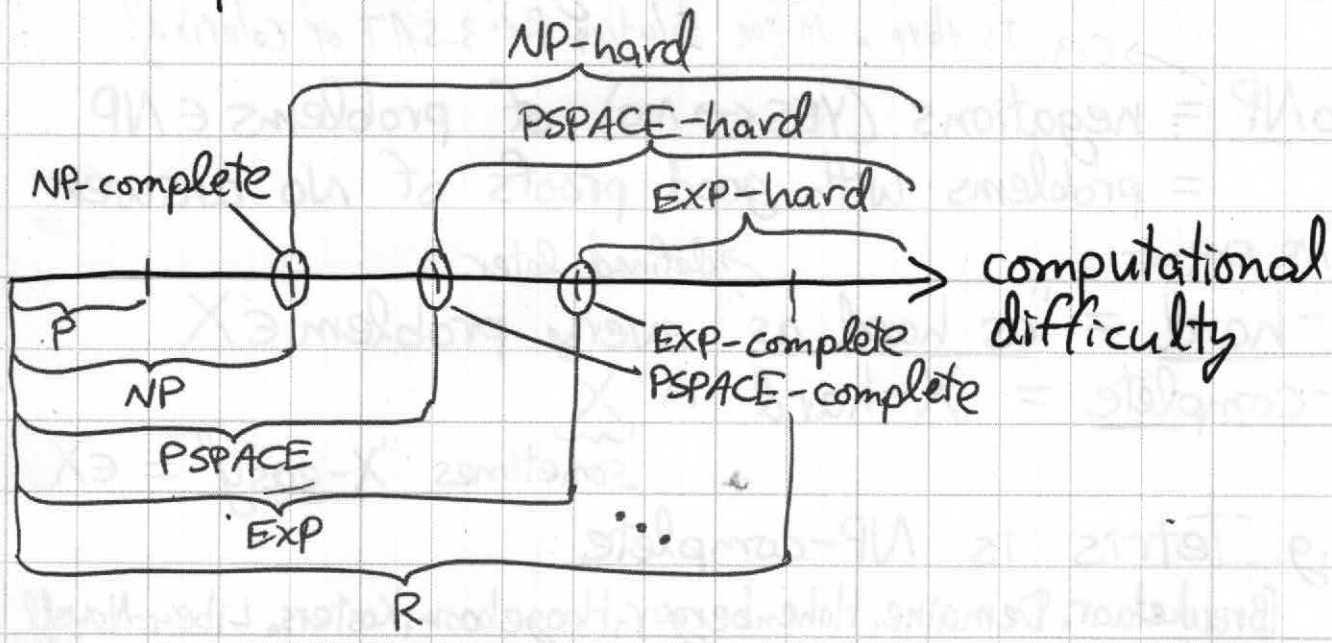
X-hard = "as hard as" (defined later) every problem ∈ X (NP, EXP, etc.)
X-complete = X-hard ∩ X
(sometimes "X-easy" = ∈X)

e.g. Tetris is NP-complete
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2004]
⇒ if P ≠ NP, then Tetris ∈ NP − P



e.g. Chess is EXP-complete ⇒ ∉ P (otherwise P = EXP)
⇒ Chess ∈ EXP − NP if NP ≠ EXP (also open)

PSPACE = {problems solvable in polynomial space}
- $\subseteq$ EXP: only exponentially many states
- $\supseteq$ NP: simulate all executions, take running OR
- open whether either is strict



e.g. Rush Hour is PSPACE-complete [Flake & Baum 2002]
$\Rightarrow \notin P$ if $P \neq NP$ or $NP \neq PSPACE$
(since $NP \subseteq PSPACE$) (since $P \subseteq NP$)

Beyond exponential: (not too important)

$EXP(TIME) \subseteq EXPSPACE \subseteq 2EXP(TIME) \subseteq 2EXPSPACE \subseteq \cdots$

(2EXP(TIME): double exponential: $2^{2^{n^c}}$)

Also $L = LOGSPACE \rightarrow O(\lg n)$ bits of space!

$EXP \subsetneq 2EXP \subsetneq \cdots$ ← time & space hierarchy theorems

$L \subsetneq PSPACE \subsetneq EXPSPACE \subsetneq 2EXPSPACE \subsetneq \cdots$ ↵

Nondeterministic:
- NPSPACE = PSPACE [Savitch 1970] (very useful!) (↗ in general, space bound squares)
- NEXP, N2EXP, ...: analogs of NP

↳ In a sense, if you can check in pspace validity of a solution then you can run over all solutions as well in PSPACE.

What does "as hard as" mean?

Reduction (from A to B) = poly-time algorithm (other constraints possible) to convert
- an instance of A to an instance of B

such that solution to A = solution to B

⇒ if can solve B then can solve A

B ∈ P, B ∈ NP, ⋮ ⟹ A ∈ P, A ∈ NP, ⋮

⇒ B is **at least as hard as** A
(A is a special case of B)

- this is a "one-call" reduction [Karp]
- "multi-call" reduction [Turing] also possible:
  solve A using an oracle that solves B
  - doesn't help much for problems we consider

Examples from algorithms:
- unweighted shortest paths → weighted ($w=1$)
- min-product path → min-sum path (lg)
- longest path → shortest path (negate)
- min-weight k-step path → min-weight path
  (k copies of graph + links between adj. layers)

Almost all hardness proofs are by reduction **from** known hard problem **to** your problem

Important Examples of Hardness Problems:

3SAT for NP-completeness: Can you satisfy (make true) a formula like $(x_1 \text{ OR } x_2 \text{ OR } x_3)$ ← variable
AND $(x_5 \text{ OR } (\text{NOT } x_3) \text{ OR } x_4)$
AND ...
(x_3: literal; x_4: literal)
3 literals per clause

For example a game such as Super Mario Bros is NP-complete via a reduction from 3-SAT [Aloupis, Demaine, Guo 2012]

Constraint Logic for PSPACE-completeness: given directed graph with edge weights $\in \{1,2\}$, find a sequence of edge reversals to reverse a target edge, while at all times maintaining total in-weight ≥ 2 at each vertex (weight entering a vertex)
Also assume all degrees = 3

Constraint logic is a powerful tool for proving hardness of games & puzzles.
e.g. for Rush Hour game [Flake & Baum '02; Hearn & Demaine '02] [Demaine and Hearn '08, CCC] proved it is PSPACE-complete.

Note that it is easy why it is PSPACE: since PSPACE = NPSPACE by Savitch's theorem, we only need to say if we are give a solution, we can check in PSPACE, which is easy since we can maintain only the current state.