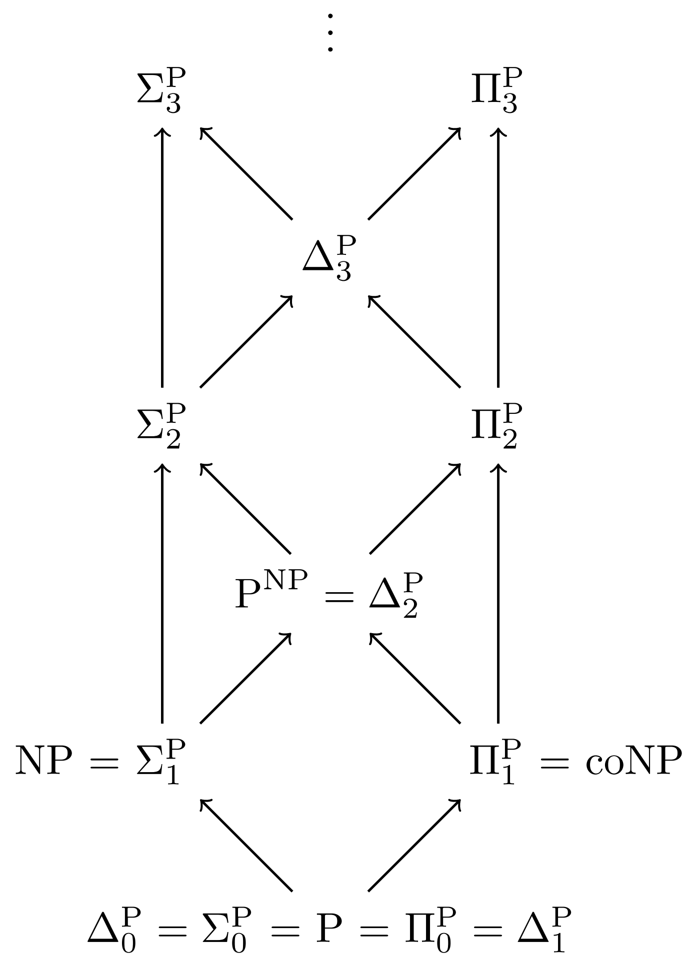
An Abridged Guide to P, NP and Some Things in Between

Nicholas LaRacuente



A sample of what's out there...

- **PSPACE** – polynomial *writeable memory*
- ***#P*** – *counting* solutions to polynomial or NP problems
- **Co-NP** – confirming that an NP problem has no solution
- **NP Hard** – at least as hard as anything in NP
- **NP Complete** – NPH & NP
- **FNP** – polynomial time to find a particular answer, such as the minimum time solution for a traveling salesman
- **NP** – polynomial time for a *non-deterministic Turing machine / checkable* in polynomial time
- **BQP** – polynomial time for a *quantum computer*
- **NP Intermediate** – in NP, but not NP Complete
- **P** – polynomial *time*
- **NL** – logarithmic *writeable memory* for a *non-deterministic* Turing machine

Rough
Order
Of *Suspected*
Difficulty

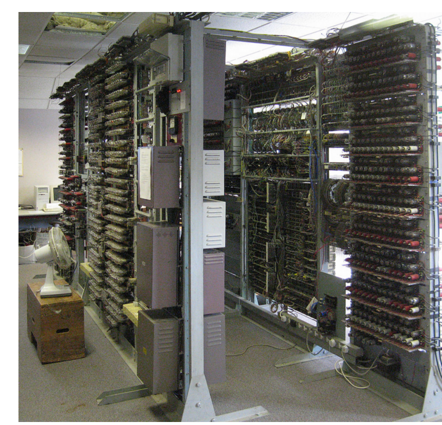
Church-Turing Thesis: *Robust* Complexity Classes

- Asymptotic scaling is invariant to changes in classical computer architecture.
- *Quantum* computers, *non-deterministic* machines & *oracles* may change the rules.

  See: Boson Sampling, Relativising Proofs

By HorsePunchKid - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=9976340

https://en.wikipedia.org/wiki/Laptop#/media/File:Aluminium_MacBook.png



By MaltaGC - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4629274

Common Points of Confusion

- NP Hard vs. NP (!=)
- BQP vs. NP (probably !=)
- FNP vs. NP

    (apples & oranges)

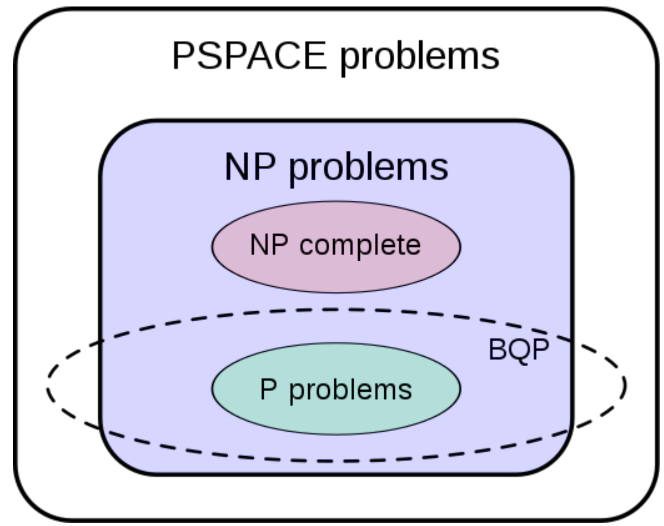


Image by User Mike1024 - Drawn by User:Mike1024This vector image was created with Inkscape., Public Domain, https://commons.wikimedia.org/w/index.php?curid=1676927

113 proofs (and counting) that P=NP, P!=NP, and the question is undecidable.

https://www.win.tue.nl/~gwoegi/P-versus-NP.htm

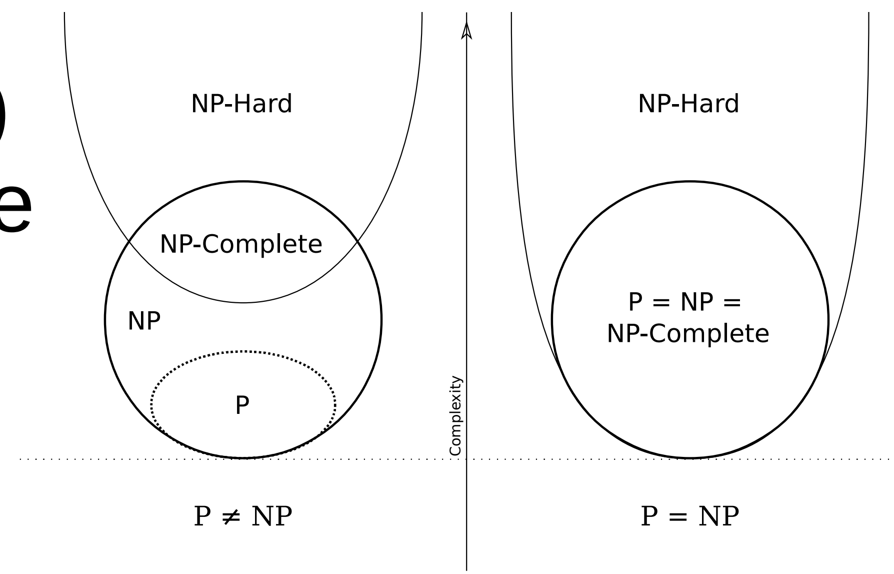


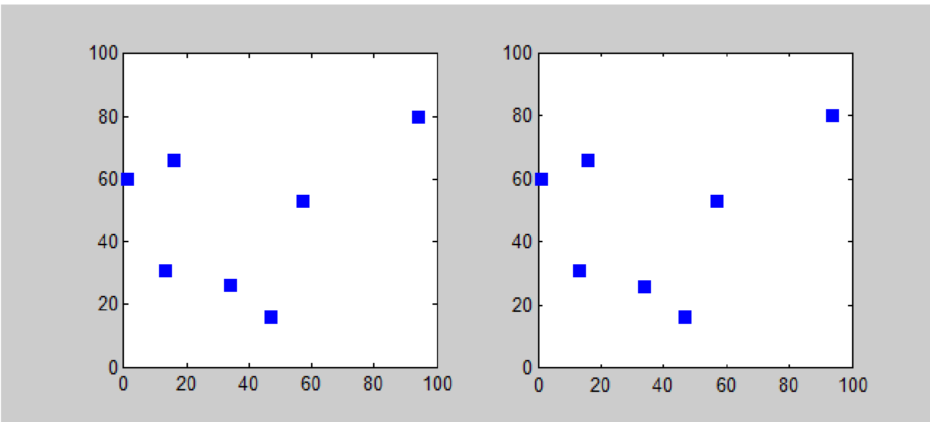
Image by Behnam Esfahbod, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3532181

Why/How NP?

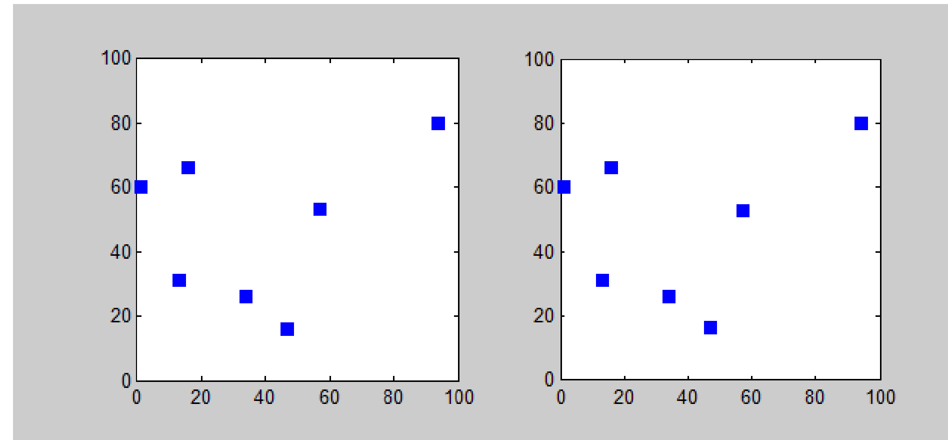
- *Non-deterministic Turing Machine* – a computer program that may take all paths for each execution branch, decide *after full execution* which to keep.
- May use execution branches to write down all of the *exponentially many* possible *certificates,* or checkable solutions to an NP problem, check in P time, then select them.
- Equivalently, lucky machine always guesses well.
- These are EXP time to simulate directly.

Traveling Salesman: Famously NPC

Brute Force Algorithm



Branch & Bound Algorithm



How to Solve?

Basic: brute force

Smart: “branch & bound” or dynamic programming – prune obviously bad pieces of solutions to reduce search space

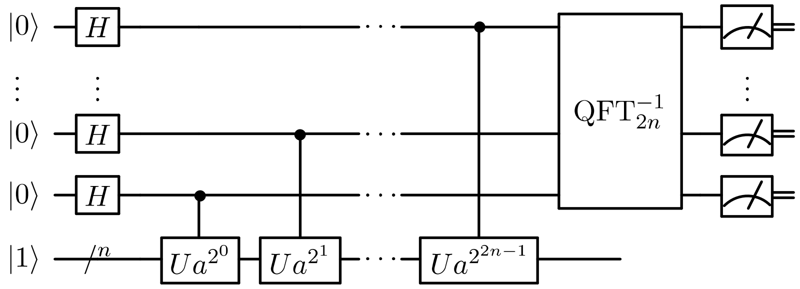
Heuristic: “Modern methods can find solutions for extremely large problems (millions of cities) within a reasonable time which are with a high probability just 2–3% away from the optimal solution.”

Ant Colony Heuristic

Quote from Wikipedia.

Integer Factors: Probably Not NPC!

- Given an n-bit integer *x*, find the factorization.
- Agrawal–Kayal–Saxena primality test: deciding whether *x* is prime is in P! doi:10.4007/annals.2004.160.781
- Shor's algorithm: factoring is in BQP
- Note that this problem has a single correct certificate – most NPC probs are ambiguous, especially w/ a fixed "good enough" criterion.



Why/How NP *Complete*?
An Case of *Reduction*

1) Take a given problem in NP.

2) Write down rules for a non-deterministic Turing machine / computer program that solves it.

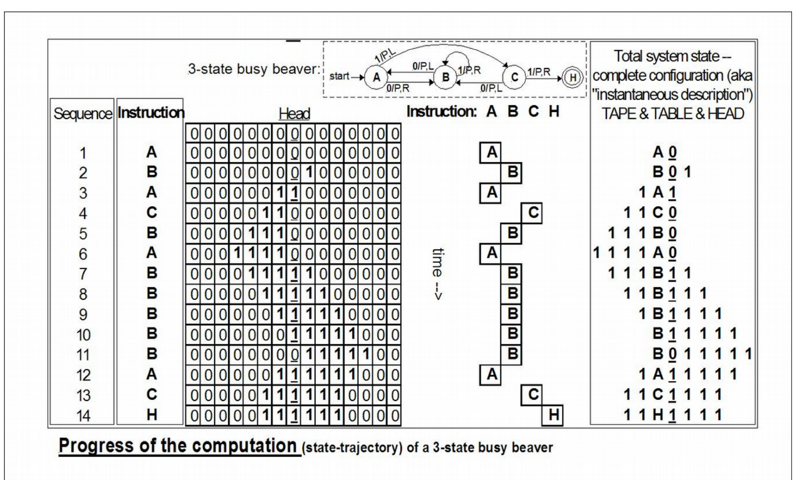
3) Convert the rules into a *bounded tiling problem*, one of the 1st known NP-Complete problems.

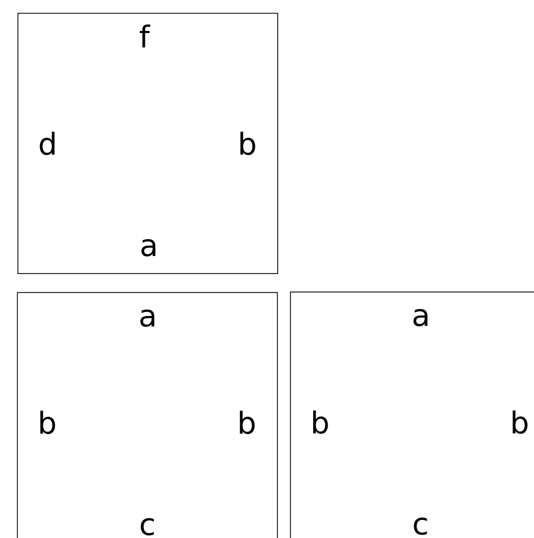
4) Anything that converts from bounded tiling is therefore also NPC. (NPC = NP & NPH)

General: Any NP problem converts in *P* time to any NPC problem. NPC != NP unless P = NP.

What is Bounded Tiling?

- **Given**: a set of 4-tuples of symbols, representing kinds of tiles. A unary-specified number, the size of a square grid to tile.
- **Rules**: Each pair of adjacent tiles must have the same symbol on their adjacent edges.
- **Use**: Let the X dimension of a grid be memory, and Y be time. Then we convert the set of transitions defined by a computer program (Turing Machine) into tile types.

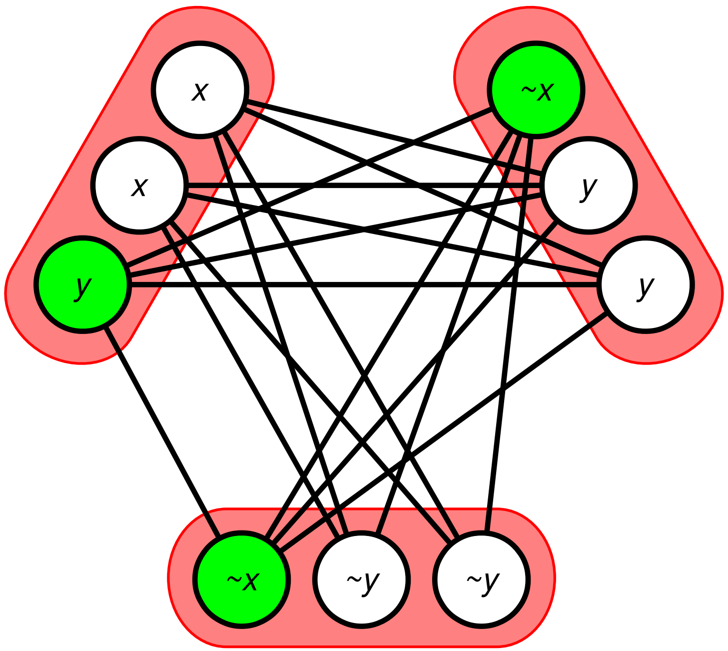




Boolean Satisfiability (SAT)

- 1st known NPC problem
- CNF (conjuctive normal form) is a *conjunction* of *disjunctions* of *vars* and *negations*
- Any SAT instance reduces to an *equisatisfiable* CNF and/or 3-CNF in P time

$$(x \lor x \lor y) \land (\neg x \lor \neg y \lor \neg y) \land (\neg x \lor y \lor y)$$



Formula from
https://en.wikipedia.org/wiki/Conjunctive_normal_form

The 3-SAT instance reduced to a clique problem. The green vertices form a 3-clique and correspond to the satisfying assignment x=FALSE, y=TRUE.

Cook's Theorem: Tiling to SAT

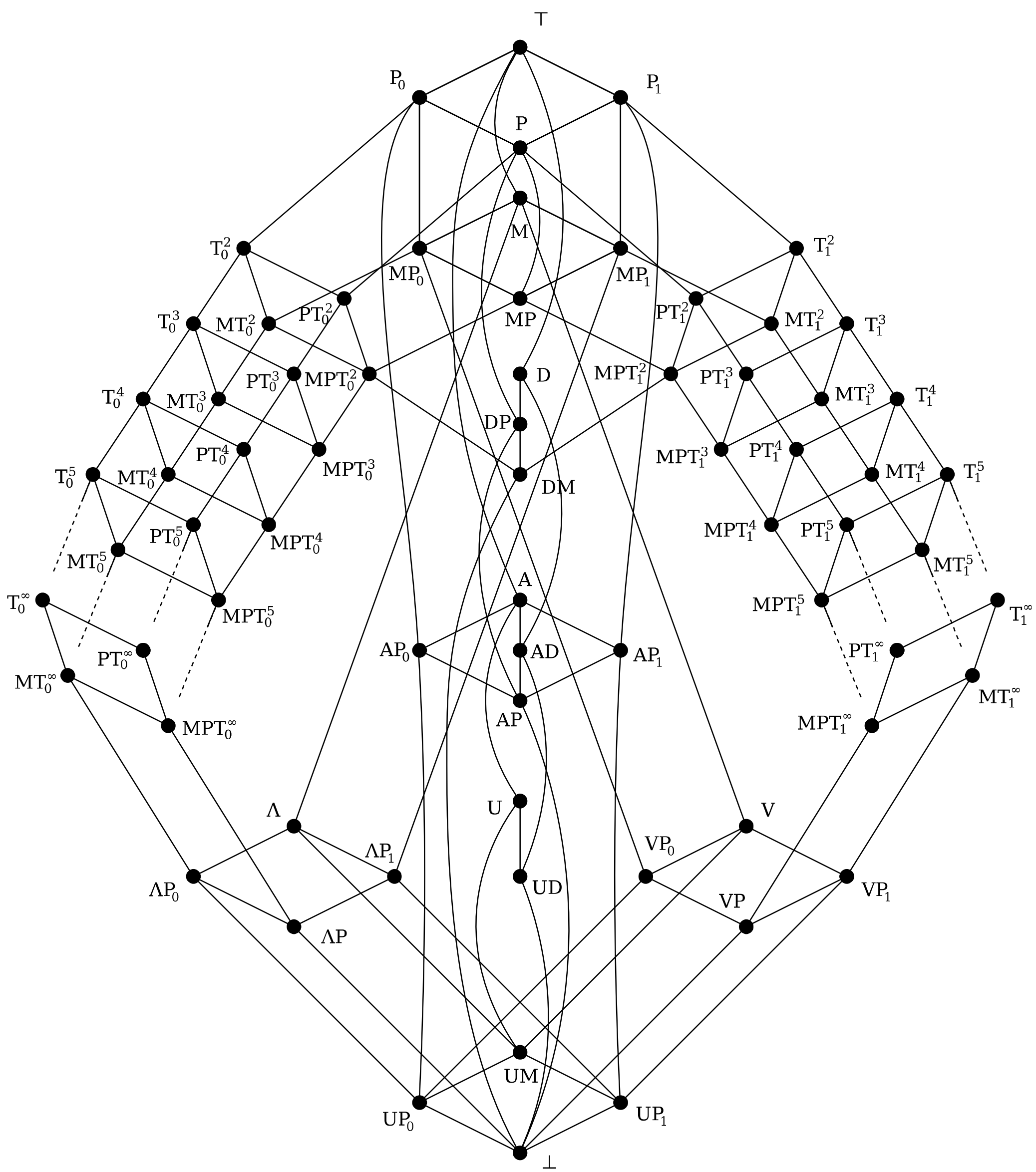
- Construct a SAT instance with a variable for each possible tile for each grid square.
- Add boolean constraints such that only 1 var is true for any grid square. We interpret this var as selecting a corresponding tile.
- Add boolean constraints to enforce adjacent tiles' same-symbol constraint.
- Solutions now correspond to tilings!
- May use auxiliary vars to convert SAT to 3CNF.

CNF Classification (Boolean Blocks)

(Now moving beyond the basic part of the presentation)

- Assume that we have available some boolean constructs: conjunction, disjunction, negation, implication, constants, etc.
- What can we make by applying & substituting constructs from this set? Some boolean functions may combine to generate others.
- Answer...

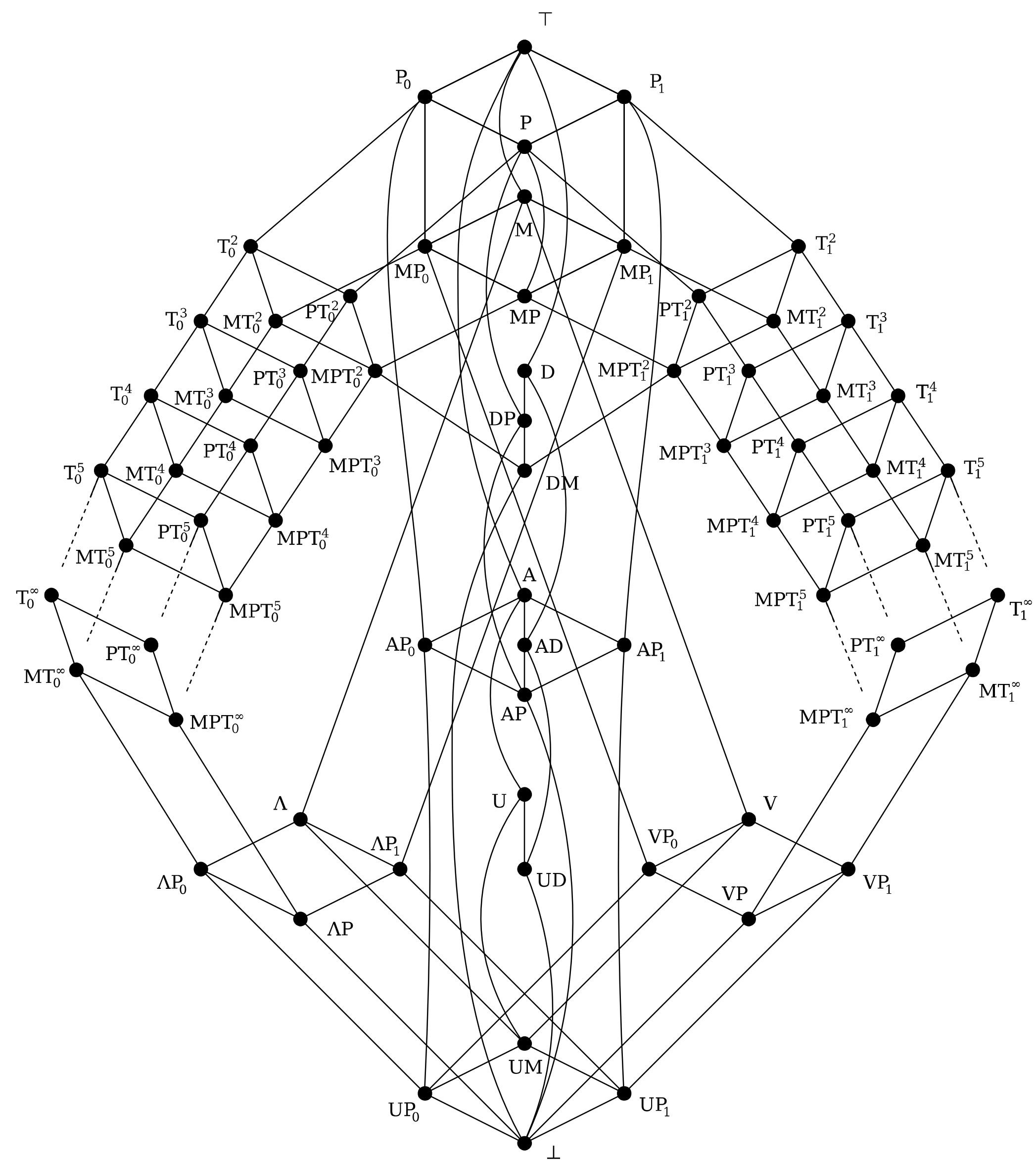
Post’s Lattice



Post's Lattice

- UP0/UP1 – constants
- VP - disjunction
- ∧P – conjunction
- …
- T – all boolean funcs

- T_0^∞ Is the smallest class that is NPC!



Galois Connection

- Q: if the boolean functions we can create are restricted, what does that say about the sets of possible solutions?

- A: We can define var-wise *polymorphisms* that, given 0-3 complete solutions, generate another satisfying solution *if the lattice class is sufficiently restrictive*.

  The existence of *non-unary polymorphisms* implies that the boolean formulae in that lattice class are always solvable in P.

Non-unary Polymorphisms

- The Constant 0/1 – trivially satisfiable by the solution that is all 1s or 0s
- The binary V/∧ - these are Horn/antiHorn CNF formulae, in which each clause contains only one positive/negated literal. It is possible to find a minimal/maximal solution in P time.
- The operation majority(x,y,z) – corresponds to 2SAT, which is *NL Complete* (and in P).
- The operations minority(x,y,z) – equivalent to xor, which allows linear algebraic solution.

Schaefer’s Dichotomy Theorem

- If a CNF has a non-unary polymorphism, it’s P

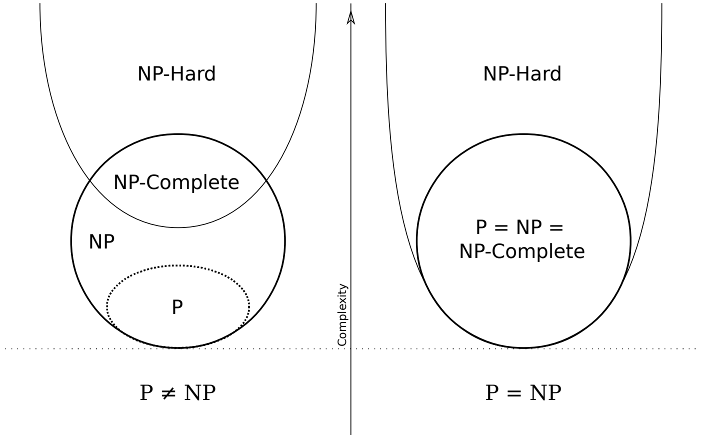
  If not, it’s NP *Complete.*

- Why? Go back to post’s lattice. Take out all the classes that satisfy any non-unary polymorphism (and anything that’s not a CNF). What we’re left with are the CNF classes that contain T_0^∞.

- So CNFs are either P or NPH! We also now have a programmatic way to decide this!

Consequences

- Difference between P and NPC for CNF formulae is an *algebraic* structure.
- There are no NP CNF formulae that are not either also in P or in NPH. Probably no BQP.
- 2CNF/3CNF distinction is especially poignant – 1 more var in clause goes from P to NPC.



Sources & Further Reading

- “A Rendevouz of Logic, Complexity and Algebra.” Hubie Chen. 2006
- “Playing With Boolean Blocks...” Bohler, Creignou, Reith & Vollmer. 2003.

Further topics (not covered today)…

- Geometric Complexity Theory
- “Statistical mechanics methods and phase transitions in optimization problems,” Martin, Monasson & Zecchina