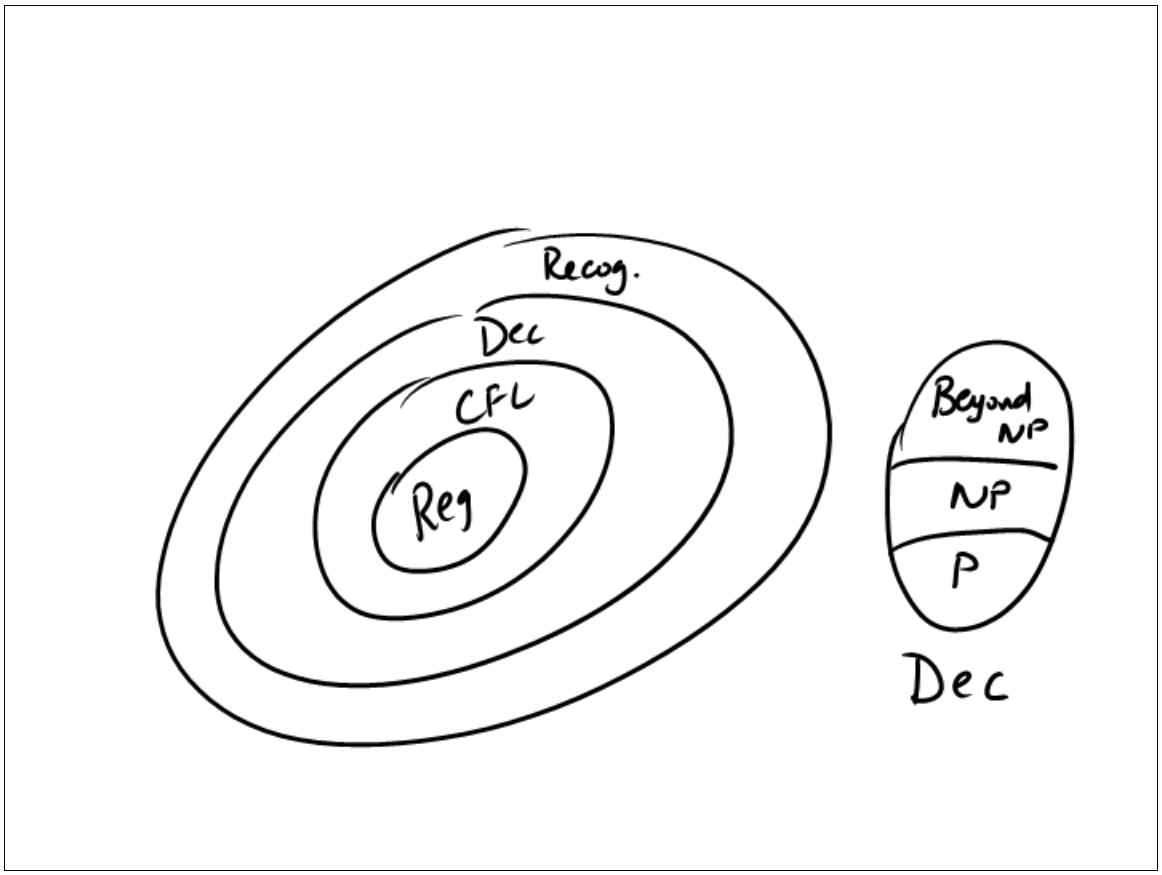
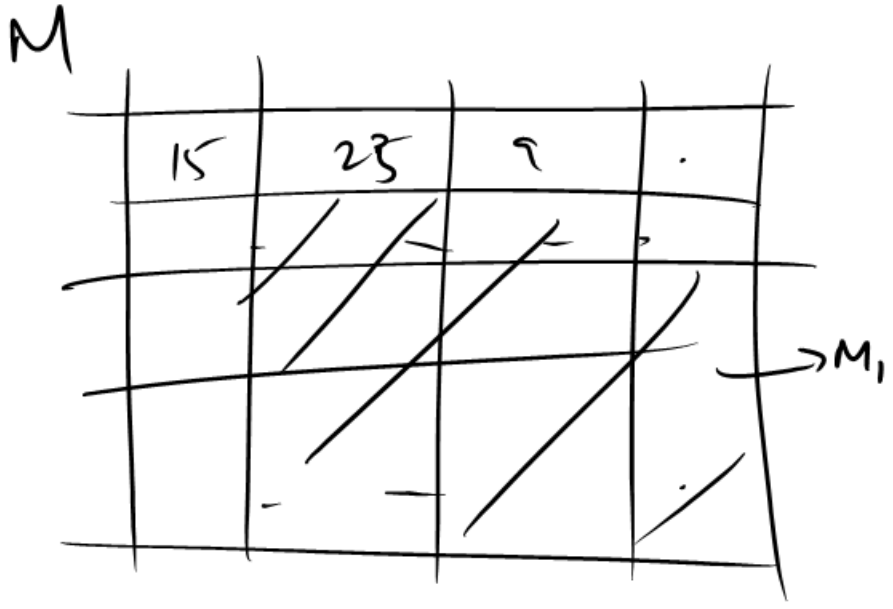
Complexity theory

- P & NP
- NP-completeness
- Cook-Levin theorem


Recog.
Dec
CFL
Reg
Beyond NP
NP
P
Dec



Determinant "15 M_1"

Alg I: "co-factor expansion" Time ($O(n!)$)

Gaussian elimination: $O(n^3)$...

$O(2^n)$ is way worse than a $O(n^3)$ algm.

Polynomial time : Solvable fast in real life.

↳ Most "natural" computation models don't differ what problems in P-time.

RAM ↪ TM
t ↪ t^2

P : The class of problems (languages) that can be decided by Turing machines working in time $O(n^k)$ for some fixed k.

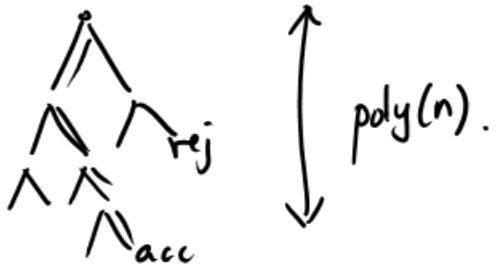
$$P = \bigcup_{k \geq 0} \text{TIME}(n^k)$$

Problems that can be solved in P-time

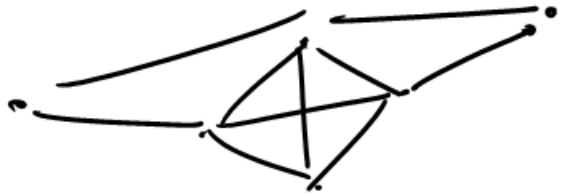
- $s \to t$ reachability in graphs. $\in P$ linear time
- CFL membership $\in P$ $O(n^3)$ time
- Sorting $\in P$ $O(n \log n)$ time ... with comparison model.
- Substring searching (grep)

Cook-Levin & NP

NP - class of problems (languages) that can be solved by nondet TM in poly time.



Given a graph $G=(V,E)$
& a number k
does G have a clique of size k?



clique of size 4
no clique of size 5.

$CLIQUE = \{ \langle G=(V,E), k \rangle \mid G$ has a clique of size $k \}$.

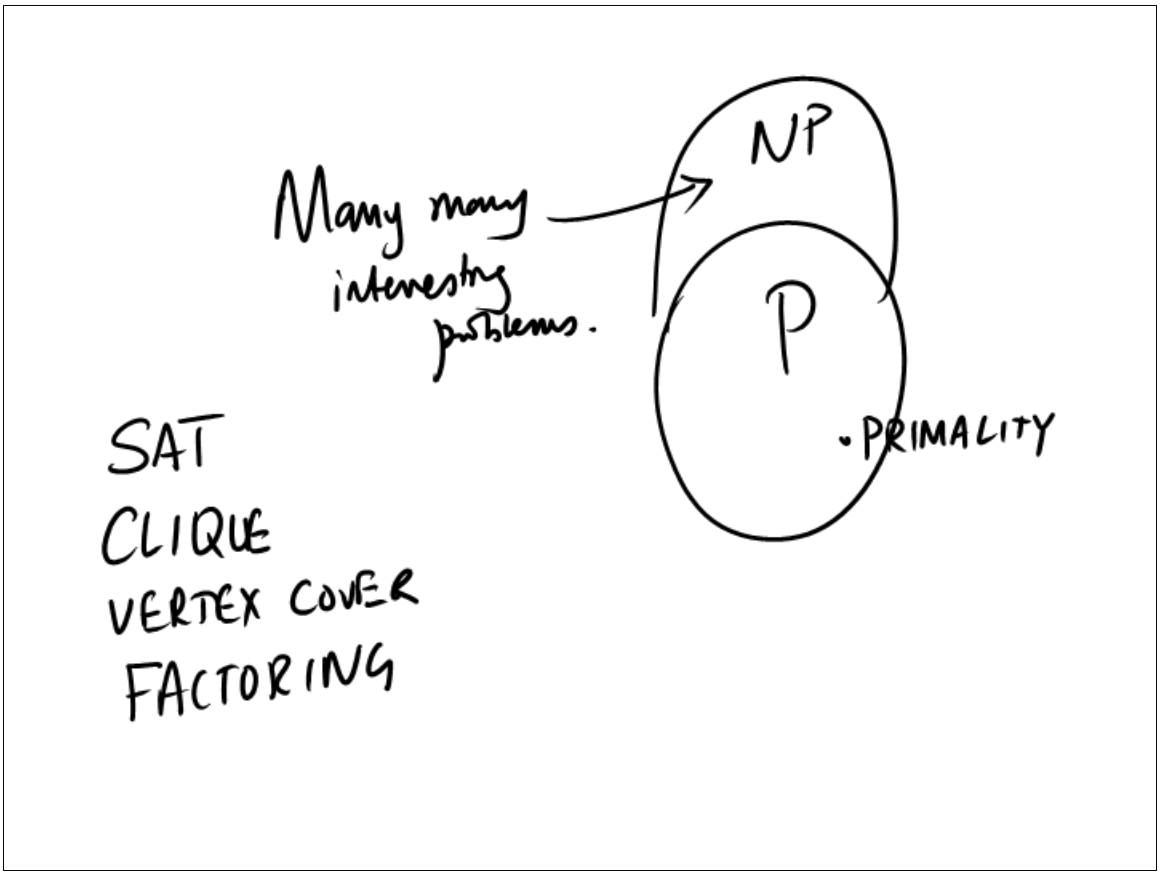
Nondet TM can solve this in time $O(n \cdot k^2)$

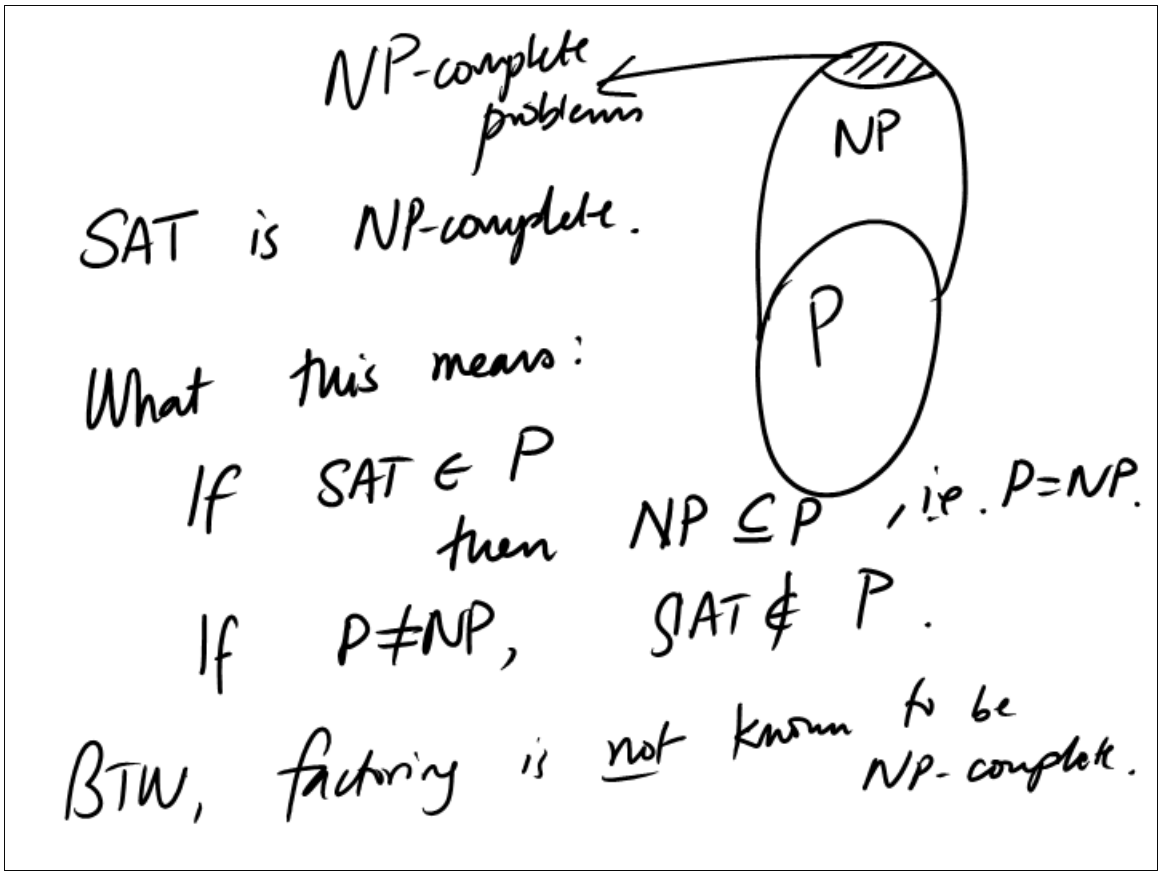
An NTM can "guess" a witness and verify whether the witness implies that $w \in L$ in poly-time.

ch

NTM for CLIQUE $(G=(V,E), k)$

- Let $S = \emptyset$; $i = 0$; $j = 0$
- while $(i < k+1)$ {
  consider j'th vertex ;
  nondet. decide {
  throw v_j into S
  i++;
  OR
  skip
  }
  }
- If $|S| = k$, check if S forms a clique. If it does, accept else reject


Many many interesting problems.
NP
P
• PRIMALITY
SAT
CLIQUE
VERTEX COVER
FACTORING

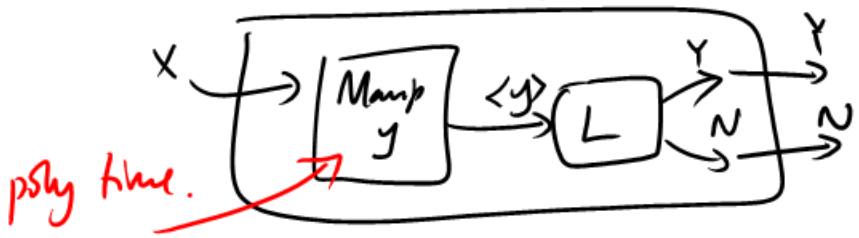

NP-complete problems
NP
P
SAT is NP-complete.
What this means:
If SAT ∈ P
then NP ⊆ P, i.e. P=NP.
If P≠NP, SAT ∉ P.
BTW, factoring is not known to be NP-complete.

Practical.

If you were given a problem, if you could show it to be NP-complete, your boss would be happy with a slow algorithm.

A language / problem is in NP if there is a NTM that solves it in poly-time.

A language L is NP-hard if every problem in NP reduces to L in poly-time.



L is NP-complete if

a) L is in NP &

b) L is NP-hard

SAT

Boolean formula : $\wedge$, $\vee$, $\neg$

$$(x \wedge y) \vee (z \wedge x \wedge \neg y) \vee ((x \wedge y) \vee (y \wedge z))$$

$$SAT = \{ \varphi \in \text{BoolForm} \mid \varphi \text{ is satisfiable} \}$$

φ is satisfiable if there is some values for the vars in φ that makes φ evaluate to T.

Why is SAT in NP?

NP machine:

- "Guess" values of $T/\bot$ for each variable.
- Evaluate formula φ on these values (inside-out, evaluating sub-exp). ... this takes only P-time.
- If φ evaluates to T, accept else reject.

SAT is NP-hard.

I.e. every problem L in NP poly-time reduces to SAT.

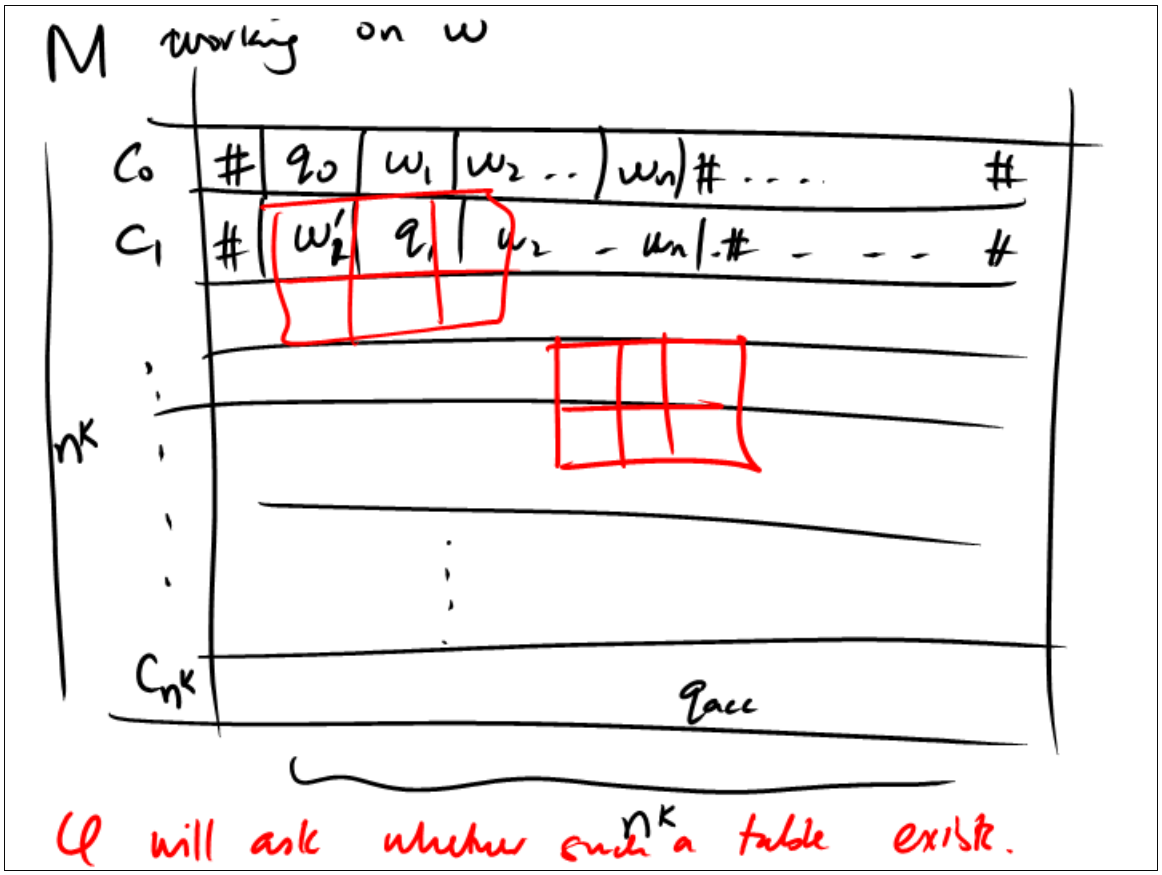
Let L be in NP

Then there is a NTM working in time $O(n^k)$ that decides L.

Given M, w
↳ NTM working in $O(n^k)$ time.

I will construct a Bool formula φ.
s.t M acc w iff φ is satisfiable.

φ must be poly in $|w|$,
and constructed in poly-time.


M working on w
C_0 # q_0 w_1 w_2 .. w_n # .... #
C_1 # w_1' q_1 w_2 - w_n . # - - - #
n^k
C_{n^k}
q_{acc}
Q will ask whether such a table exists.

c_{ij} can have a symbol in $T \cup Q$.

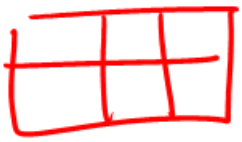
$x_{i,j,s} = T \Leftrightarrow c_{ij}$ has symbol s

φ : φ_{valid} — every config has only one state & the rest are T.

$\wedge$ φ_{start} — first config is $\#q_0 w \# \ldots \#$

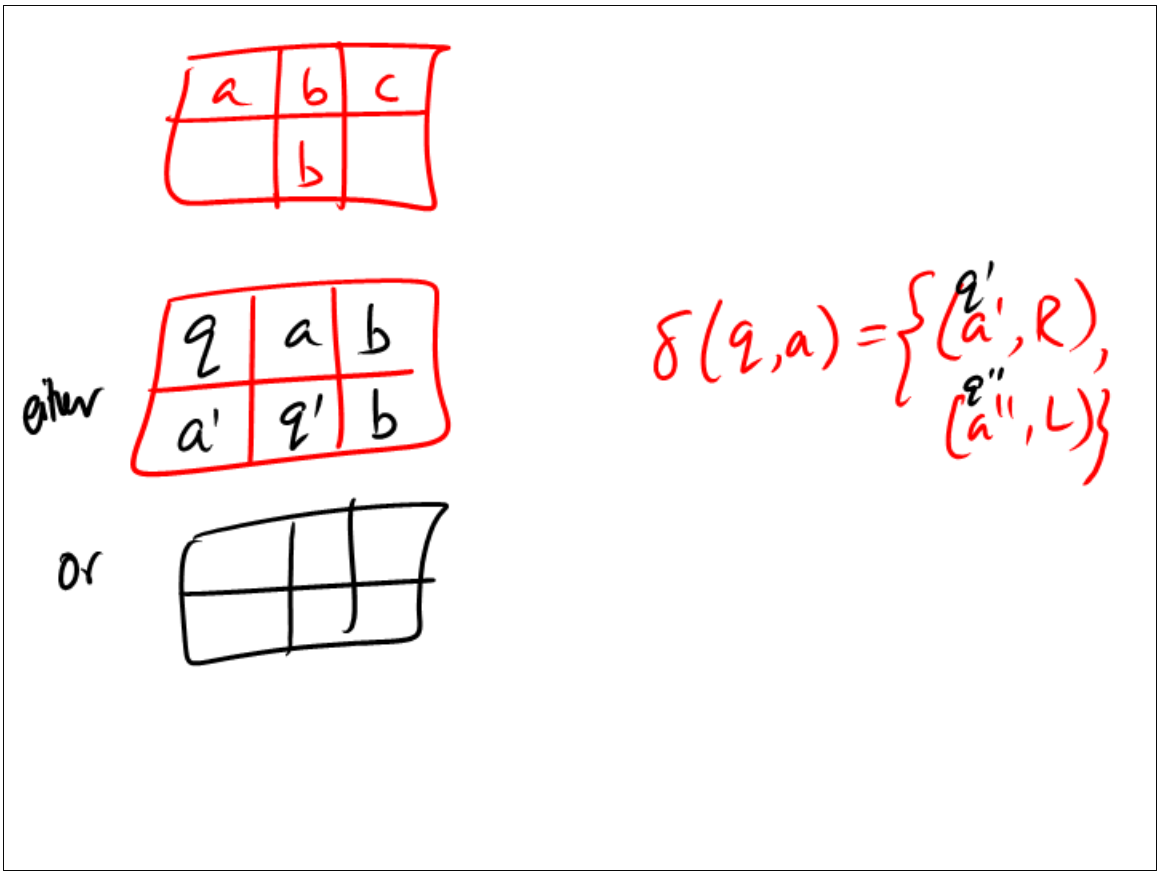
$\wedge$ φ_{end} — last config must be an acc. config.

$\wedge$ φ_{move} — look at each window



$|\varphi|$ polynomial in n ($O(n^{2k})$)

So SAT is NP-complete.


a b c
b
either
q a b
a' q' b
or
δ(q,a) = {(q' a', R), (q'' a'', L)}