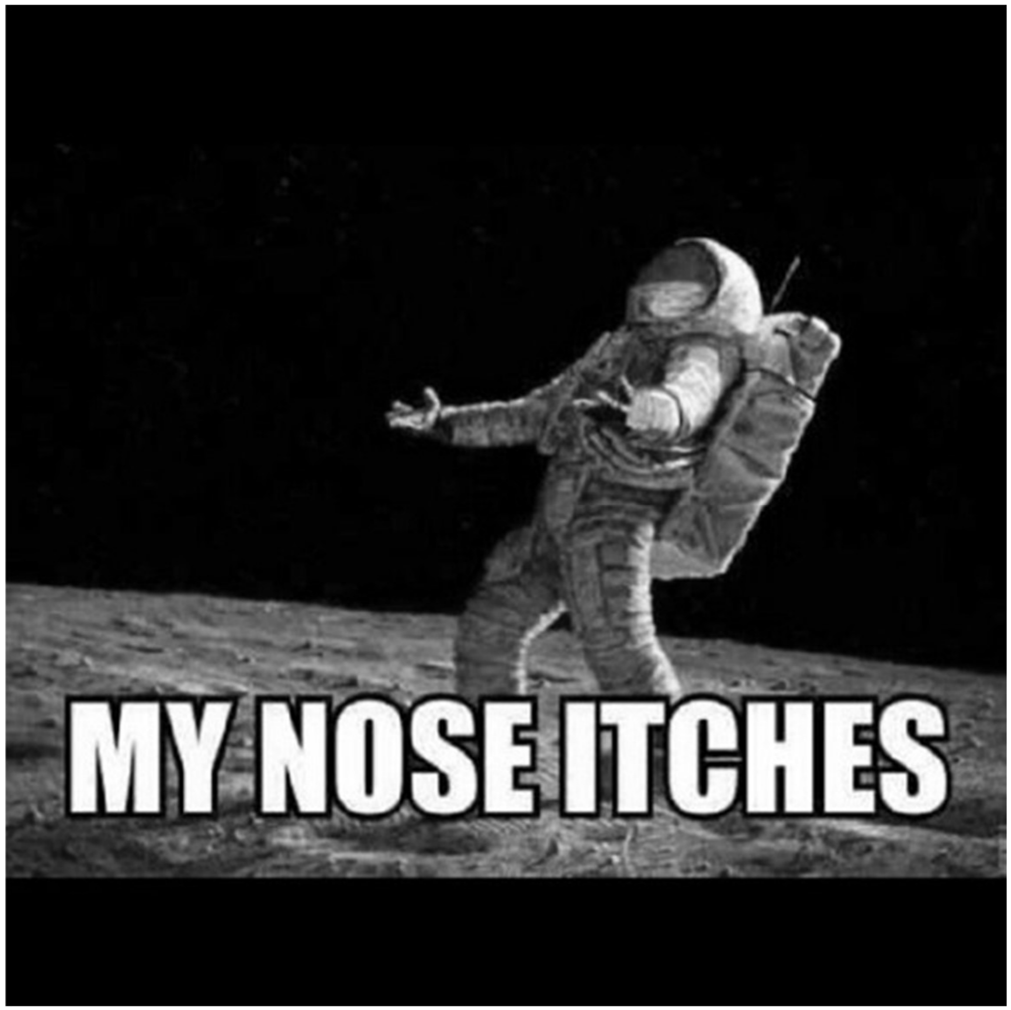
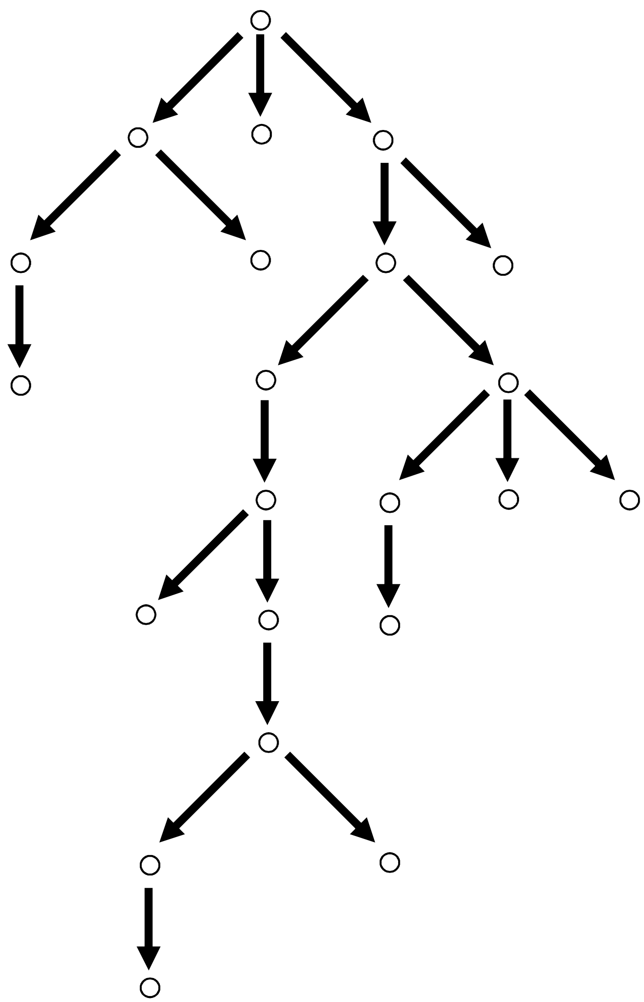
Space Complexity



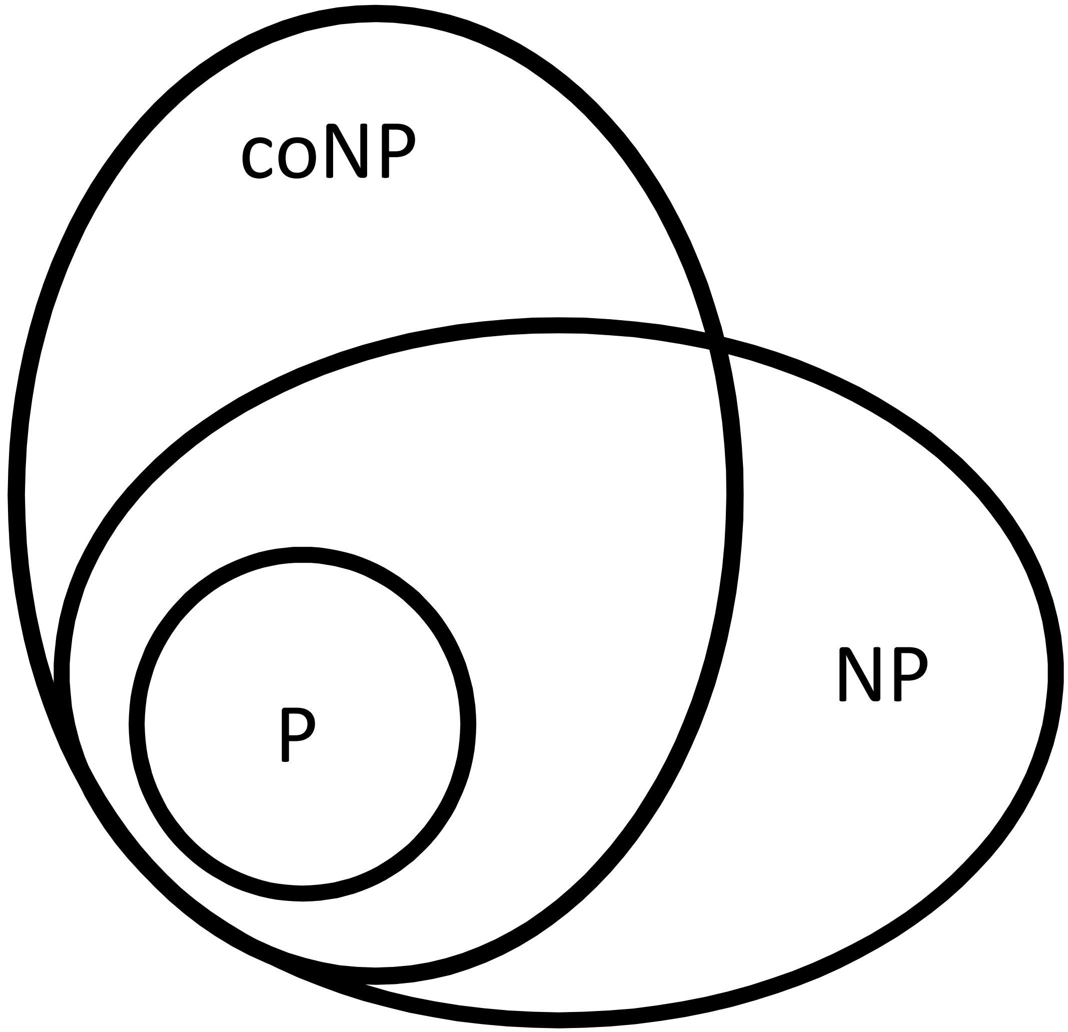
Definition: coNP = { L | $\neg$L $\in$ NP }

What does a coNP computation look like?



In NP algorithms, we can use a "guess" instruction in pseudocode:
Guess string y of |x|^k length…
and the machine accepts if some y leads to an accept state

In coNP algorithms, we can use a "try all" instruction:
Try all strings y of |x|^k length…
and the machine accepts if every y leads to an accept state


coNP
NP
P

Definition: A language B is coNP-complete if

1. **B $\in$ coNP**

2. **For every A in coNP, there is a polynomial-time reduction from A to B**

(B is coNP-hard)

UNSAT = { ϕ | ϕ is a Boolean formula and *no* variable assignment satisfies ϕ }

Theorem: UNSAT is coNP-complete

Proof: UNSAT $\in$ coNP because $\neg$UNSAT $\approx$ SAT

(2) UNSAT is coNP-hard:

Let A $\in$ coNP. We show A $\leq_P$ UNSAT

On input w, transform w into a formula ϕ using the Cook-Levin Theorem and an NP machine N for $\neg$A

w $\in \neg$A $\Rightarrow$ $\phi \in$ SAT

$\boxed{\text{w} \notin \text{A} \Rightarrow \phi \notin \text{UNSAT}}$

w $\notin \neg$ A $\Rightarrow$ $\phi \notin$ SAT

$\boxed{\text{w} \in \text{A} \Rightarrow \phi \in \text{UNSAT}}$

UNSAT = { ϕ | ϕ is a Boolean formula and *no* variable assignment satisfies ϕ }

Theorem: UNSAT is coNP-complete

TAUTOLOGY = { ϕ | ϕ is a Boolean formula and *every* variable assignment satisfies ϕ }

= {ϕ | $\neg\phi \in$ UNSAT}

Theorem: TAUTOLOGY is coNP-complete

(1) TAUTOLOGY $\in$ coNP (already shown)

(2) TAUTOLOGY is coNP-hard:

UNSAT $\leq_P$ TAUTOLOGY:

Given formula ϕ, output $\neg\phi$

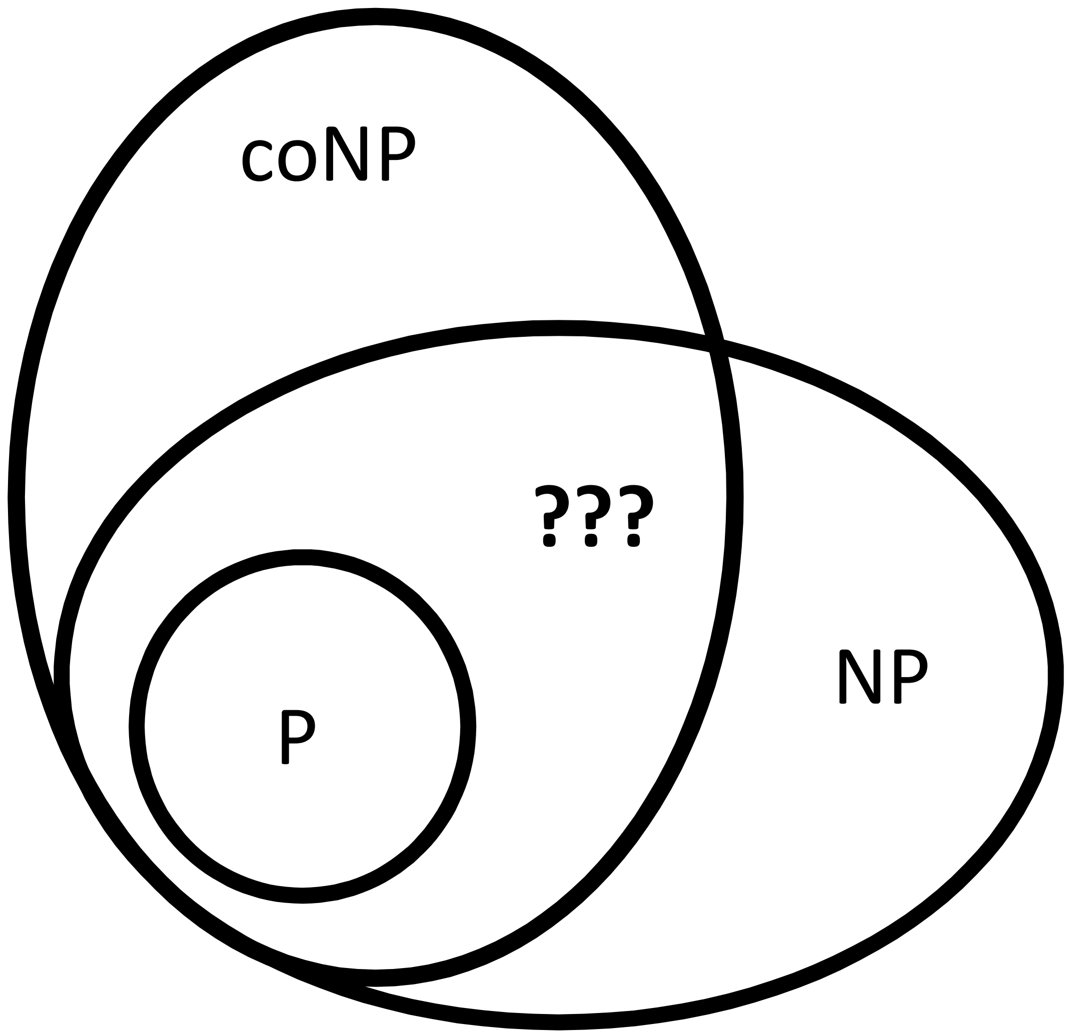
Every NP-complete problem has a coNP-complete counterpart

NP-complete problems:

SAT, 3SAT, CLIQUE, VC, SUBSET-SUM, …

coNP-complete problems:

UNSAT, TAUTOLOGY, NOCLIQUE, …


coNP
???
P
NP

Is P = NP $\cap$ coNP?

THIS IS AN OPEN QUESTION!

An Interesting Problem in NP $\cap$ coNP

FACTORING
= { (m, n) | m > n > 1 are integers,
there is a prime factor p of m where n ≤ p < m }

If FACTORING $\in$ P, then we could break most public-key cryptography currently in use!

Theorem: FACTORING $\in$ NP $\cap$ coNP

To show that FACTORING $\in$ NP $\cap$ coNP, we'll use

PRIMES = {n| n is a prime integer}

PRIMES is in P

Manindra Agrawal, Neeraj Kayal and Nitin Saxena

Ann. of Math. Volume 160, Number 2 (2004), 781-793.

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

FACTORING

= { (m, n) | m, n > 1 are integers,

there is a prime factor p of m where $n \leq p < m$ }

Theorem: FACTORING $\in$ NP $\cap$ coNP

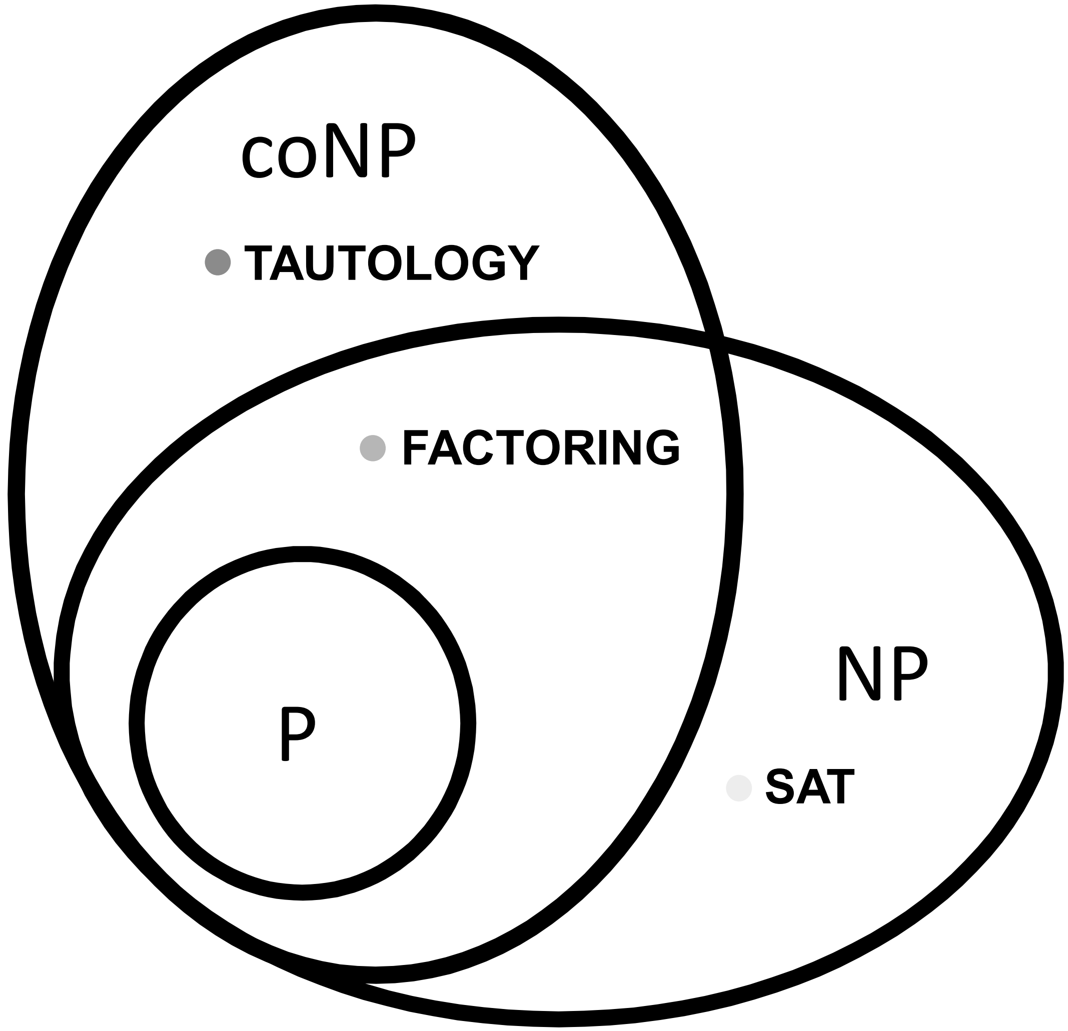
Proof:

The prime factorization $p_1^{e1} \ldots p_k^{ek}$ of m can be used to efficiently prove that either (m,n) is in FACTORING or (m,n) is not in FACTORING:

First *verify* each p_i is prime and $p_1^{e1} \ldots p_k^{ek} = m$

If there is a $p_i \geq n$ then (m,n) is in FACTORING

If for all i, $p_i < n$ then (m,n) is not in FACTORING


coNP
TAUTOLOGY
FACTORING
P
NP
SAT

How to Think about Oracles?

Think in terms of Turing Machine pseudocode!

An oracle Turing machine M with oracle B $\subseteq \Gamma^*$ lets you include the following kind of branching instructions:

"if (z in B) then <do something>
else <do something else>"

where z is some string defined earlier in pseudocode. By definition, the oracle TM can always check the condition (z in B) in one step

This notion makes sense even if B is not decidable!

Some Complexity Classes With Oracles

P^B = { L | L can be decided by some *polynomial-time* TM with an oracle for B }

P^{SAT} = the class of languages decidable in polynomial time with an oracle for SAT

P^{NP} = the class of languages decidable by *some* polynomial-time oracle TM with an oracle for *some* B in NP

Is $P^{SAT} \subseteq P^{NP}$?

Yes! By definition…

Is $P^{NP} \subseteq P^{SAT}$?

Yes!

Every NP language can be reduced to SAT!

For every poly-time TM M with oracle B $\in$ NP, we can simulate every query z to oracle B by reducing z to a formula ϕ in poly-time, then asking an oracle for SAT instead

P^B = { L | L can be decided by a polynomial-time TM with an oracle for B }

Suppose B is in P.

Is $P^B \subseteq P$?

Yes!

For every poly-time TM M with oracle $B \in P$, we can simulate every query z to oracle B by simply running a polynomial-time decider for B.

The resulting machine runs in polynomial time!

Is NP $\subseteq$ P^{NP}?

Yes!

Just ask the oracle for the answer!

For every L $\in$ NP define an oracle TM M^L which asks the oracle if the input is in L.

Is coNP $\subseteq$ P^{NP}?

Yes!

Again, just ask the oracle for the answer!

For every L $\in$ coNP we know $\neg$L $\in$ NP

Define an oracle TM M$^{\neg L}$ which asks the oracle if the input is in $\neg$L

***accept* if the answer is no,**

***reject* if the answer is yes**

In general, we have P^{NP} = P^{coNP}

P^{NP} = the class of languages decidable by some polynomial-time oracle TM M^B for some B in NP

Informally: P^{NP} is the class of problems you can solve in polynomial time, assuming SAT solvers work

NP^B = { L | L can be decided by a polynomial-time nondeterministic TM with an oracle for B }

$coNP^B$ = { L | L can be decided by a poly-time co-nondeterministic TM with an oracle for B }

Is $NP = NP^{NP}$?

Is $coNP^{NP} = NP^{NP}$?

THESE ARE OPEN QUESTIONS!

It is believed that the answers are NO

Logic Minimization is in $\mathrm{coNP}^{\mathrm{NP}}$

Two Boolean formulas ϕ and ψ over the variables $x_1,\ldots,x_n$ are equivalent if they have the same value on every assignment to the variables

Are x and $x \vee x$ equivalent? Yes

Are x and $x \vee \neg x$ equivalent? No

Are $(x \vee \neg y) \wedge \neg(\neg x \wedge y)$ and $x \vee \neg y$ equivalent? Yes

A Boolean formula ϕ is minimal if no smaller formula is equivalent to ϕ

MIN-FORMULA = $\{ \phi \mid \phi \text{ is minimal} \}$

Theorem: MIN-FORMULA $\in$ coNP^{NP}

Proof:

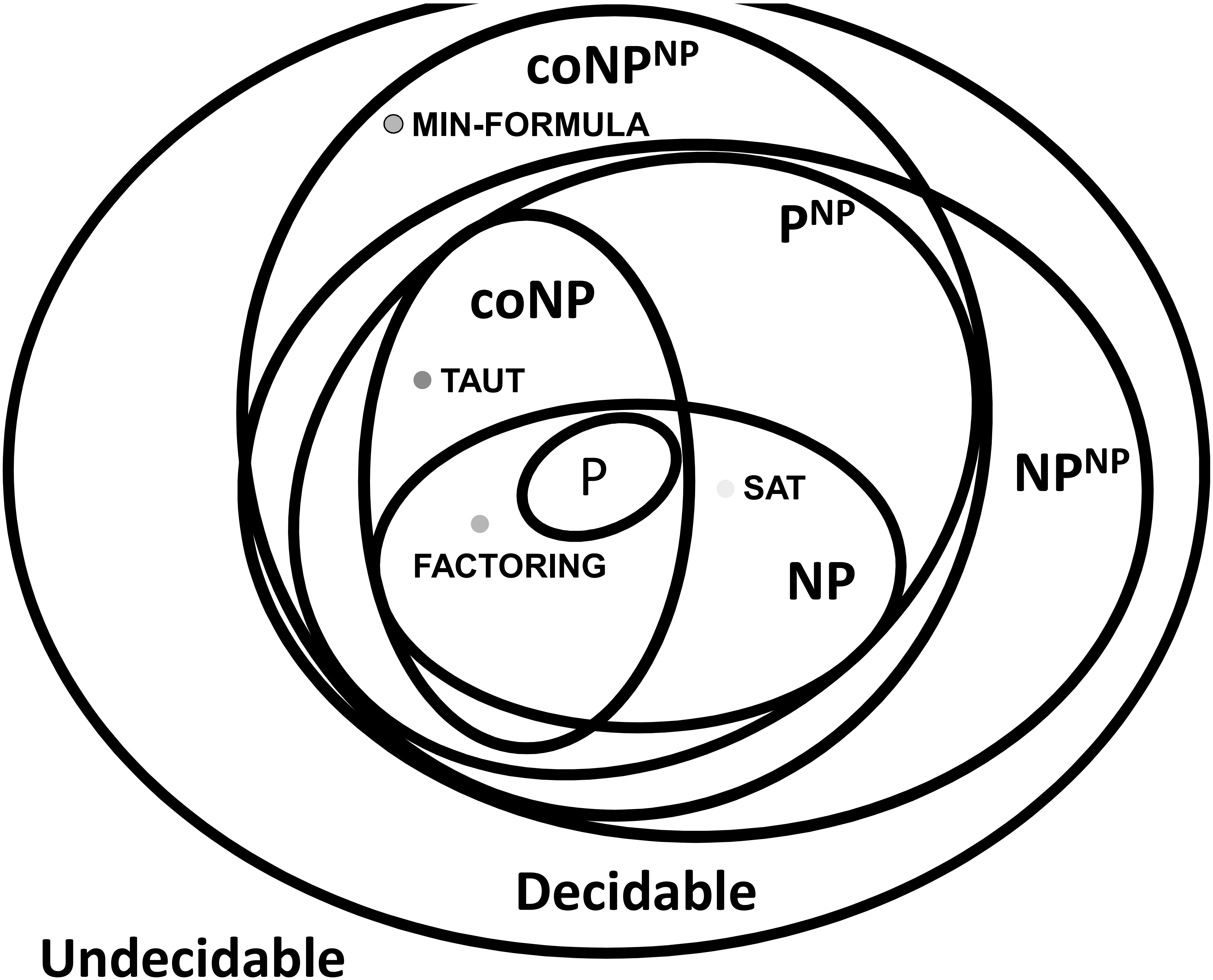
Define NEQUIV = { (ϕ, ψ) | ϕ and ψ are not equivalent }

Observation: NEQUIV $\in$ NP (Why?)

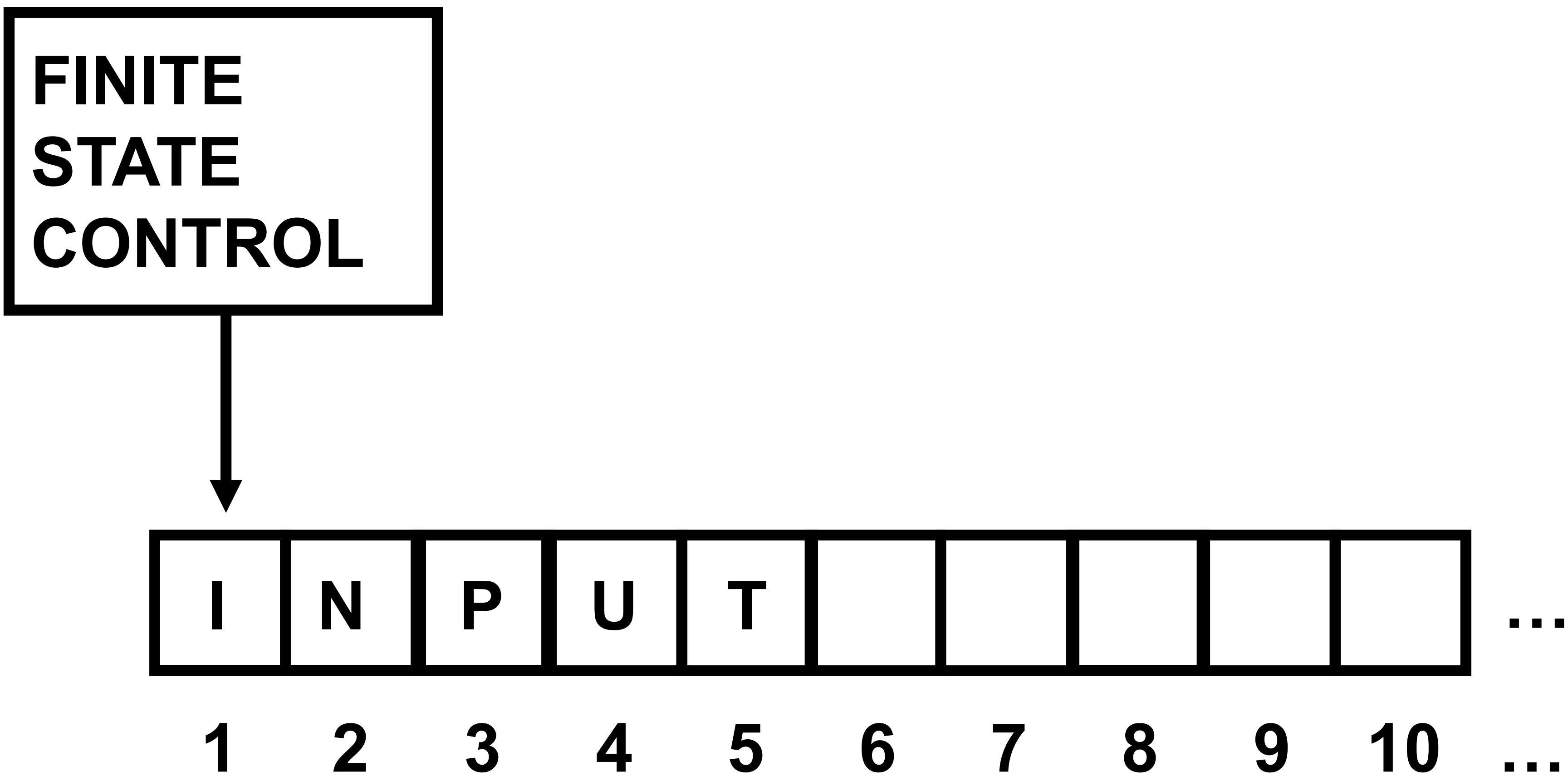
Here is a coNP^{NEQUIV} machine for MIN-FORMULA:

Given a formula ϕ,
Try all formulas **ψ smaller than ϕ:**
If (ϕ, ψ) $\in$ NEQUIV then *accept* else *reject*

MIN-FORMULA is not known to be in coNP!


coNP^NP
MIN-FORMULA
P^NP
coNP
TAUT
P
SAT
NP^NP
FACTORING
NP
Decidable
Undecidable

Measuring Space Complexity



We measure *space* complexity by looking at the largest tape index reached during the computation

Let M be a deterministic TM.

Definition: The space complexity of M is the function $S : \mathbb{N} \to \mathbb{N}$, where $S(n)$ is the largest tape index reached by M on any input of length n.

Definition: SPACE($S(n)$) =

{ L | L is decided by a Turing machine with O($S(n)$) space complexity}

Theorem: 3SAT $\in$ SPACE(n)

"Proof": Try all possible assignments to the (at most n) variables in a formula of length n. This can be done in O(n) space.

Theorem: NTIME(t(n)) is in SPACE(t(n))

"Proof": Try all possible computation paths of t(n) steps for an NTM on length-n input. This can be done in O(t(n)) space.

The class SPACE(s(n)) formalizes the class of problems solvable by computers with *bounded memory*.

Fundamental (Unanswered) Question:
How does time relate to space, in computing?

SPACE(n^2) problems could potentially take much longer than n^2 steps to solve!

Intuition: You can always re-use space, but how can you re-use time?

Time Complexity of SPACE(S(n))

Let M be a halting TM that on input x, uses S space

How many time steps can M(x) possibly take?

Is there an upper bound?

The number of time steps is at most the total number of possible *configurations*!

(If a configuration repeats, the machine is looping.)

A configuration of M specifies a head position, state, and S cells of tape content. The total number of configurations is at most:

$$S\ |Q|\ |\Gamma|^S = 2^{O(S)}$$

Corollary:
Space S(n) computations can be decided in $2^{O(S(n))}$ time

$$\mathrm{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbf{N}} \mathrm{TIME}(2^{c \cdot s(n)})$$

Idea: After $2^{O(s(n))}$ time steps, a s(n)-space bounded computation must have repeated a configuration, so then it will never halt…

$$\mathbf{PSPACE} = \bigcup_{k \in \mathbf{N}} \mathbf{SPACE}(n^k)$$

$$\mathbf{EXPTIME} = \bigcup_{k \in \mathbf{N}} \mathbf{TIME}(2^{n^k})$$

$$\mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$$

Is P $\subseteq$ PSPACE?

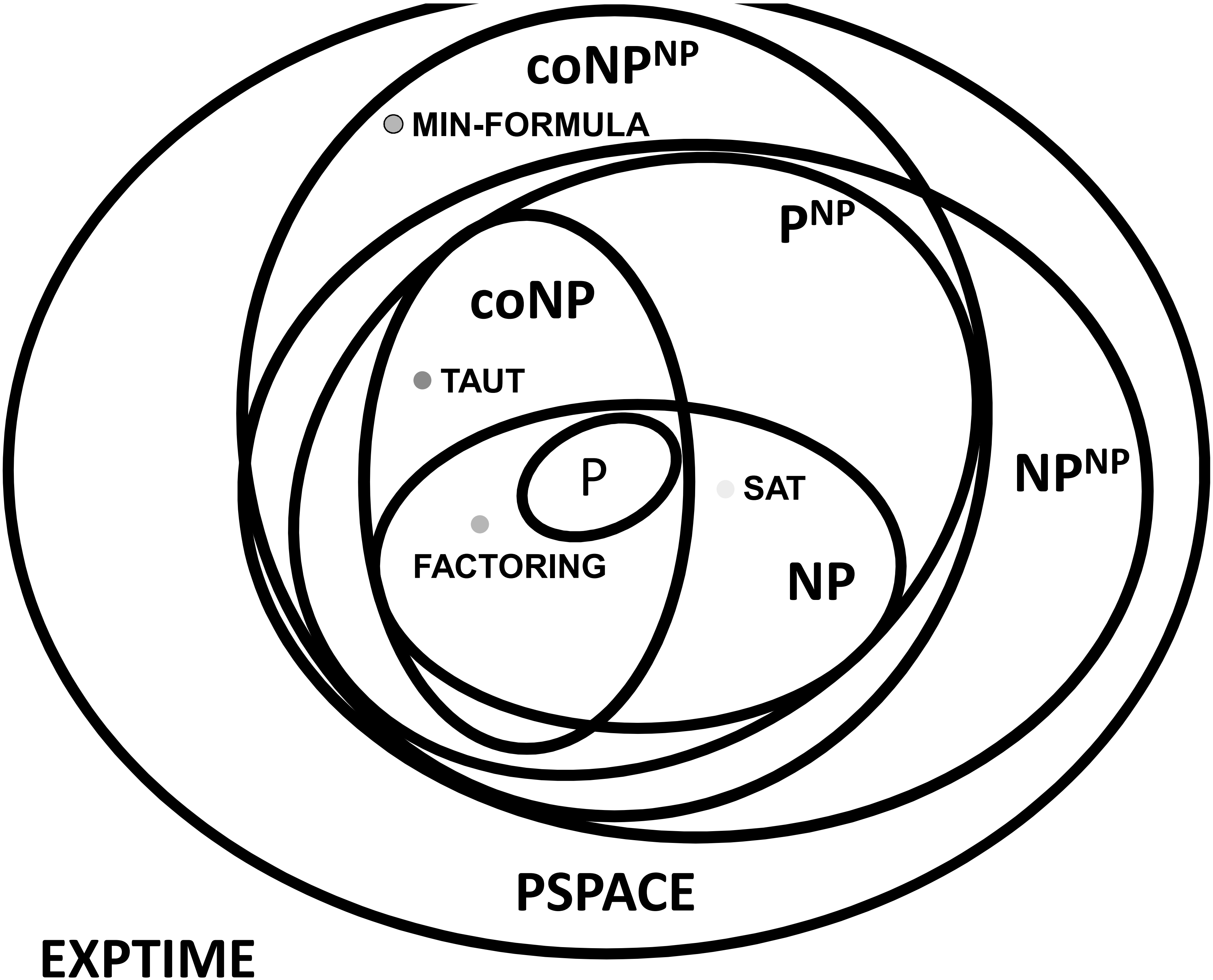
YES

Is NP $\subseteq$ PSPACE?

YES

Is $NP^{NP} \subseteq$ PSPACE?

YES


coNP^NP
MIN-FORMULA
P^NP
coNP
TAUT
P
SAT
NP^NP
FACTORING
NP
PSPACE
EXPTIME

Thank you!

For being a great class!