CS 154

Lecture 17: coNP, Oracles, Space Complexity

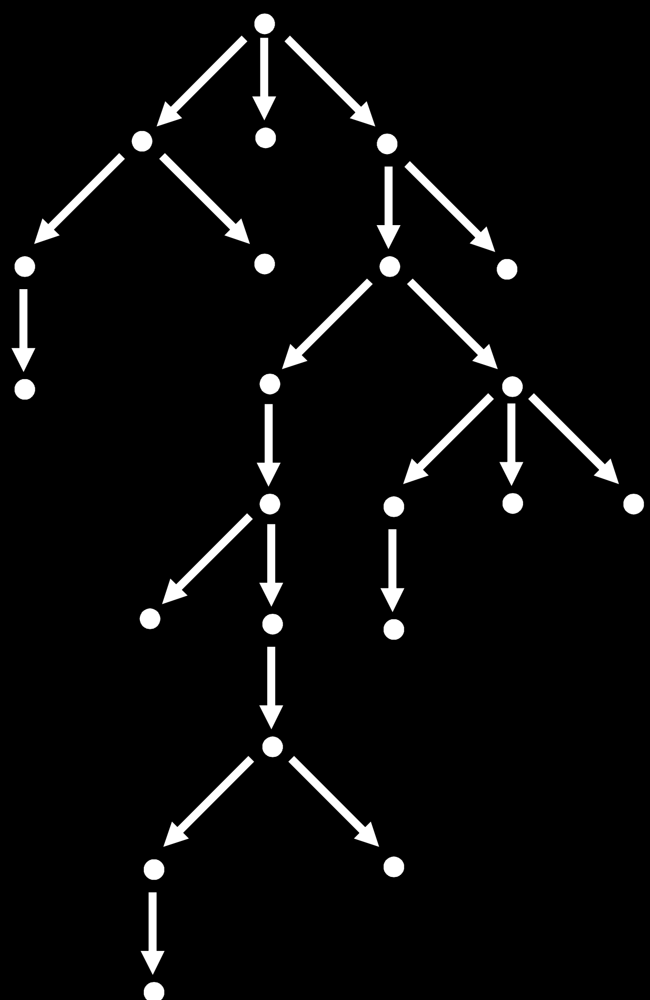
VOTE VOTE VOTE

For your favorite course on automata and complexity

Please complete the online course evaluation

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

What does a coNP computation look like?

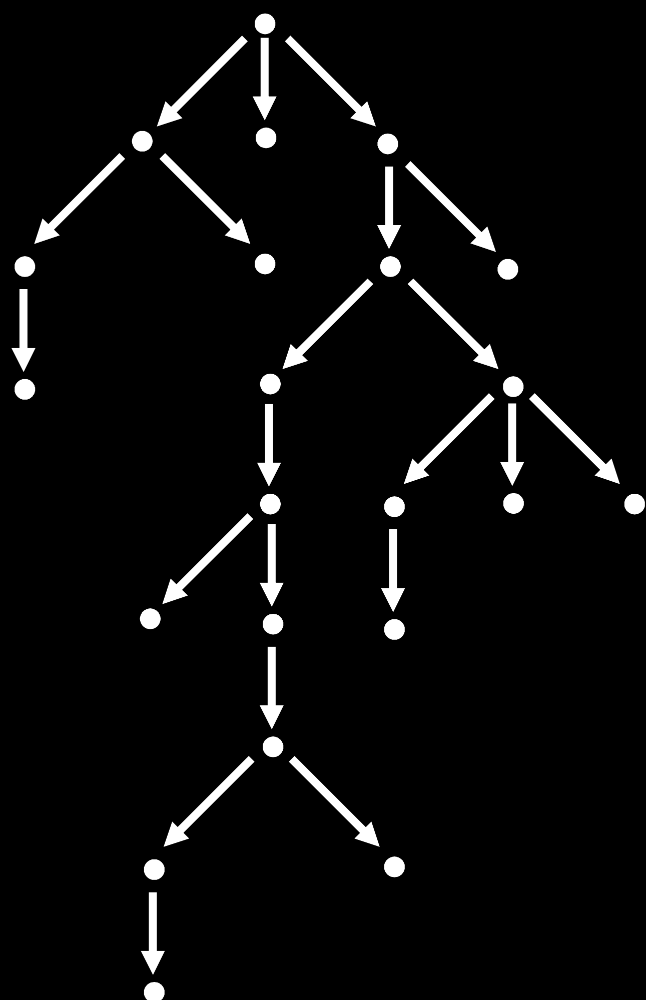


A *co-nondeterministic* machine has multiple computation paths, and has the following behavior:

- the machine **accepts** if *all paths reach* accept state
- the machine **rejects** if *at least one path* reaches reject state

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…

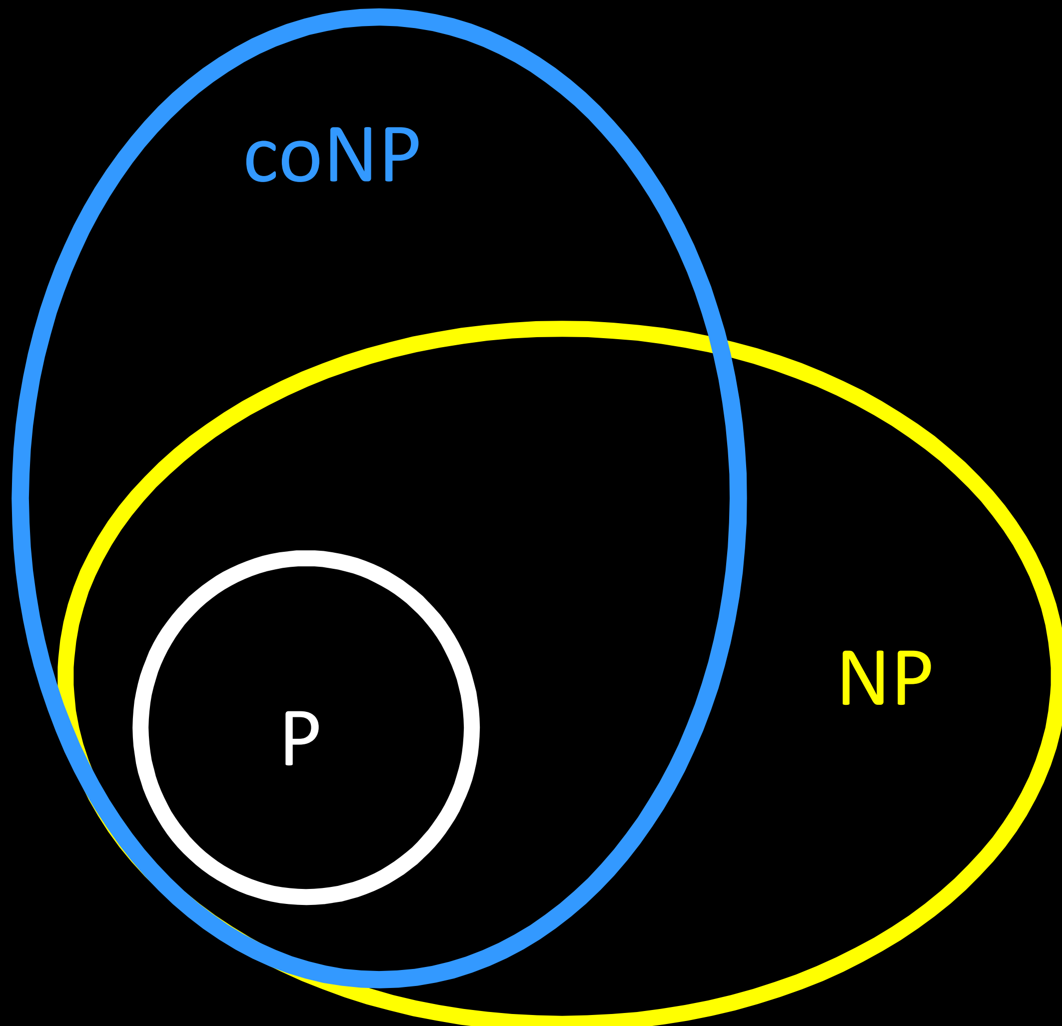
and the machine accepts if every y leads to an accept state

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

Theorem: TAUTOLOGY is in coNP

How would we write pseudocode for a coNP machine that decides TAUTOLOGY?

How would we write TAUTOLOGY as the complement of some NP language?


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 Cook-Levin via an NTM for $\neg$A

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

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

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

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

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}

TAUTOLOGY is coNP-complete

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

(2) TAUT is coNP-hard: UNSAT $\leq_P$ TAUT:

Given formula ϕ, output $\neg\phi$

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 $n \leq p < m$ }

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

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

PRIMES = {n| n is a prime integer}

Theorem (Pratt): PRIMES $\in$ NP $\cap$ coNP

PRIMES is in P

Manindra Agrawal, Neeraj Kayal and Nitin Saxena

Source: 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 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

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: (1) FACTORING $\in$ NP

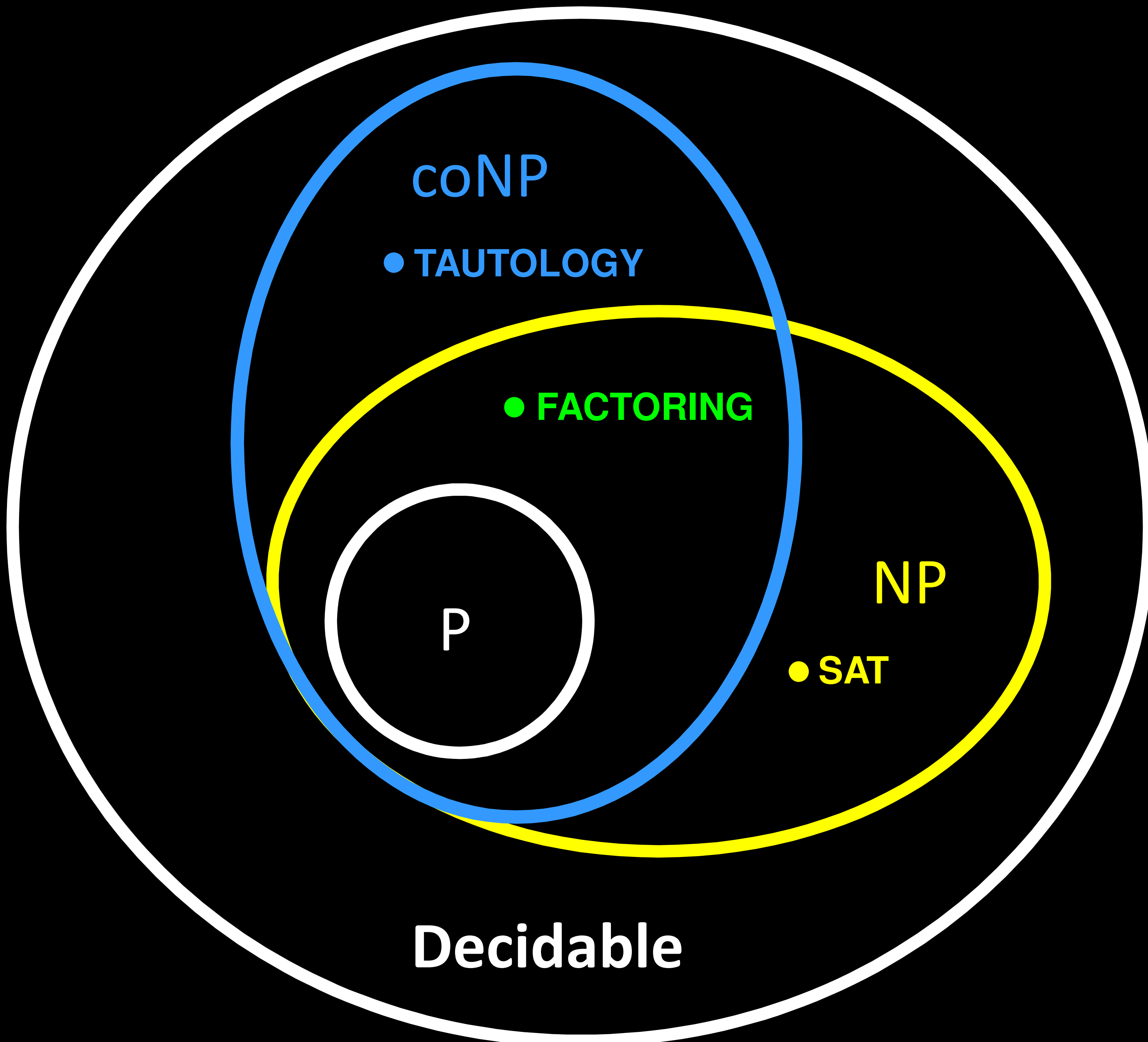
Any prime factor p of m such that $p \geq n$
is a proof that (m,n) is in FACTORING

(2) FACTORING $\in$ coNP

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

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

Then check that for all i that $p_i < n$


coNP
TAUTOLOGY
FACTORING
NP
P
SAT
Decidable

NP-complete **problems:**

SAT, 3SAT, CLIQUE, VC, SUBSET-SUM, …

coNP-complete **problems:**

UNSAT, TAUTOLOGY, NOHAMPATH, …

(NP $\cap$ coNP)-complete **problems:**

Nobody knows if they even exist!

P, NP, coNP can be defined in terms of specific machine models, and for every possible machine we can give an encoding of it.

NP $\cap$ coNP is *not* known to have a corresponding machine model!

Polynomial Time With Oracles

***We do not condone smoking. Don't do it. It's bad. Kthxbye**

Oracle Turing Machines

q_{YES}

Is formula F in SAT?

yes

A	N	P	U	T					

INFINITE TAPE

Oracle Turing Machines

An **oracle Turing machine M^B** is equipped with a set $B \subseteq \Gamma^*$ to which a TM M may ask membership queries on a special "oracle tape"
[Formally, M^B enters a special state $q_?$]

and the TM receives a query answer in one step
[Formally, the transition function on $q_?$ is defined in terms of the *entire oracle tape*:
if the string y written on the oracle tape is in B, then state $q_?$ is changed to q_{YES}, otherwise q_{NO}]

This notion makes sense even when
M runs in polynomial time and B is not in P!

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 w to oracle B by reducing w to a SAT instance in polytime 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 w 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, this is the class of problems you can solve in polynomial time, assuming SAT solvers work

A typical problem in P^{NP}:

FIRST-SAT = { (ϕ, i) | $\phi \in$ SAT and the ith bit of the lexicographically first SAT assignment of ϕ is 1}

Using polynomially many calls to SAT, we can *compute* the lex. first satisfying assignment!

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 $coNP^{NP}$

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 EQUIV = { (ϕ, ψ) | ϕ and ψ are equivalent }

Observation: EQUIV $\in$ coNP   (Why?)

So EQUIV can be decided with an oracle for SAT.

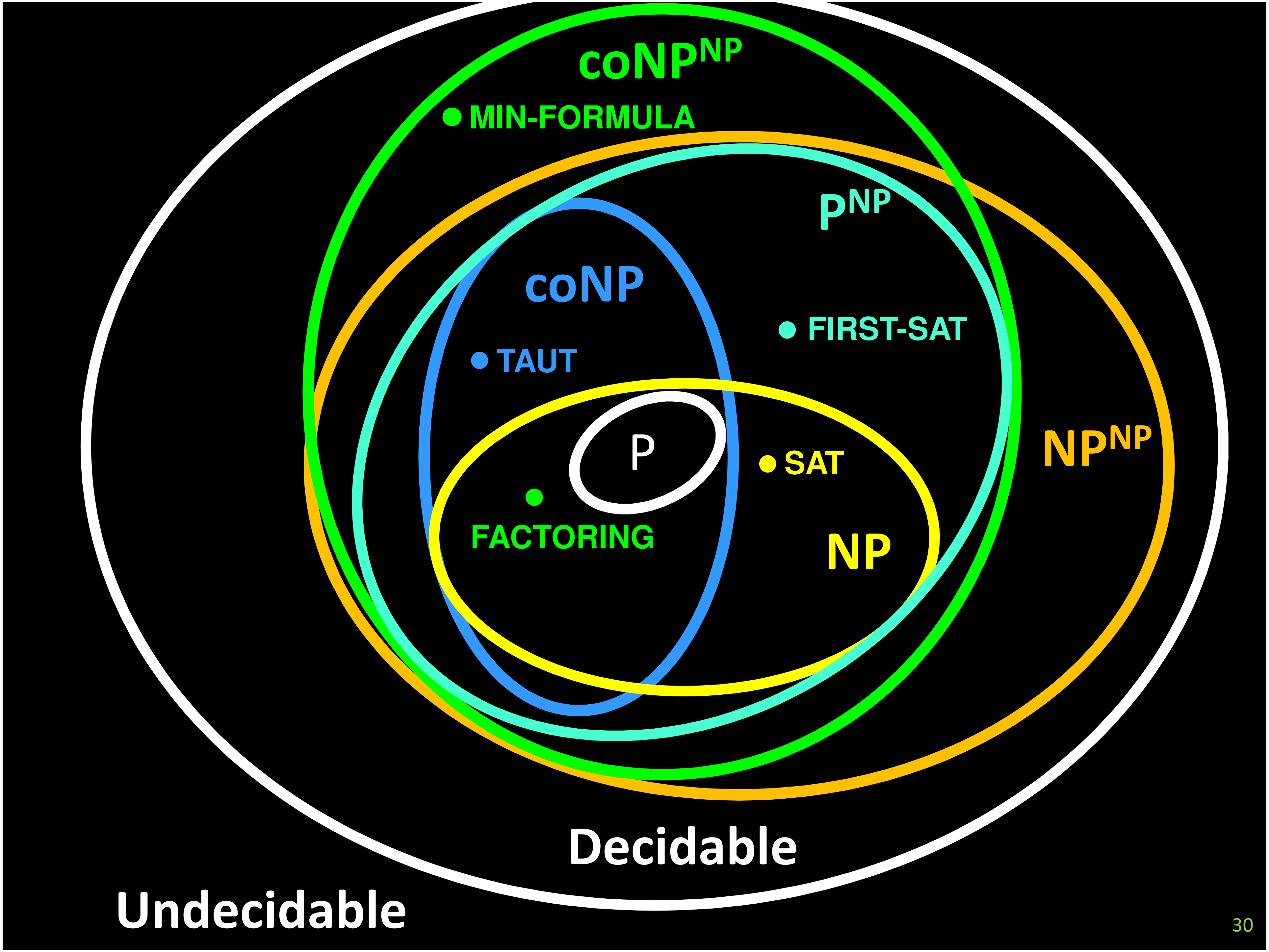
Here is a coNP^{EQUIV} machine for MIN-FORMULA:

Given a formula ϕ,

Try all formulas ψ smaller than ϕ:

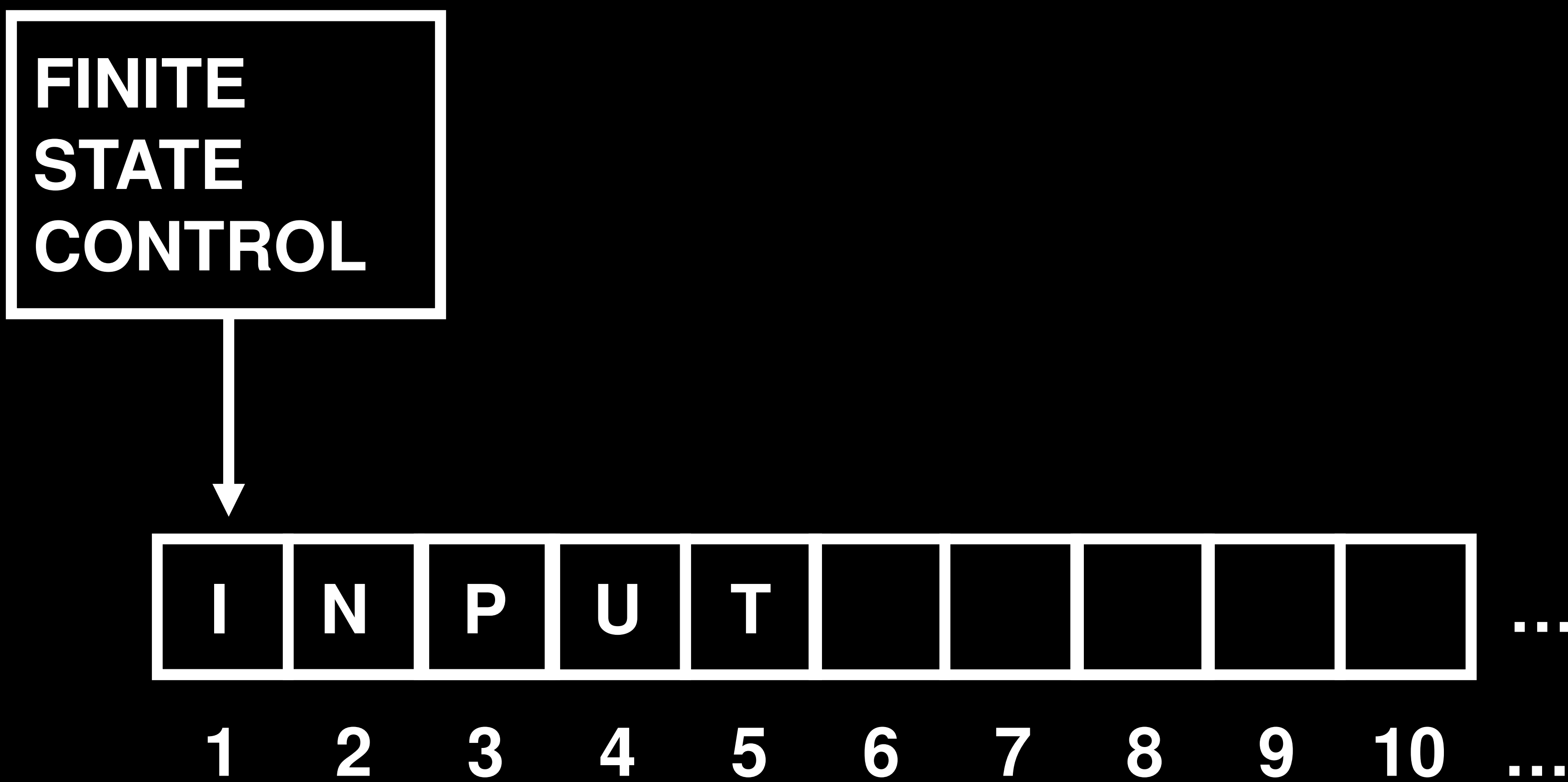
If $((\phi, \psi) \in$ EQUIV) then *reject* else *accept*

MIN-FORMULA is not known to be in coNP!


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

Space Complexity

Measuring Space Complexity



We measure *space* complexity by looking at the furthest tape cell reached during the computation

Let M be a deterministic TM.

Definition: The space complexity of M is the function $f : N \to N$, where f(n) is the furthest tape cell 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": Exhaustively checking all possible assignments to the (at most n) variables in a formula can be done in O(n) space.

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

"Proof": Exhaustively checking all possible computation paths of t(n) steps for an NTM can be done in O(t(n)) space.

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

Does this remind you of something?
Oh… right… the real world…

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?

Space Hierarchy Theorem

Intuition: If you have more *space* to work with, then you can solve strictly more problems!

Theorem: For functions s, S : N → N where s(n)/S(n) → 0

$$\mathrm{SPACE}(s(n)) \subsetneq \mathrm{SPACE}(S(n))$$

Proof IDEA: Diagonalization

Make a machine M that uses S(n) space and "does the opposite" of all s(n) space machines on at least one input

So L(M) is in SPACE(S(n)) but not SPACE(s(n))

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

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

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…

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

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

PSPACE $\subseteq$ EXPTIME