Lecture 20:
More Friends of NP,
Oracles in Complexity Theory
Definition: coNP = \{ L \mid \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 iff 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 iff every y leads to an accept state
Definition: A language B is coNP-complete if

1. $B \in \text{coNP}$

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

Can use $A \leq_P B \iff \neg A \leq_P \neg B$

to turn NP-hardness into co-NP hardness
TAUTOLOGY = \{ \phi \mid \phi \text{ is a Boolean formula and } no\text{ every variable assignment satisfies } \phi \}\}

Theorem: TAUTOLOGY is coNP-complete

\text{UNSAT} = \{ \phi \mid \phi \text{ is a Boolean formula and no variable assignment satisfies } \phi \} \}
= \{ \phi \mid \neg \phi \in \text{UNSAT} \} 

Theorem: UNSAT is coNP-complete
NP \cap \text{coNP} = \{ L \mid L \text{ and } \neg L \in \text{NP} \}

L \in \text{NP} \cap \text{coNP} \text{ means that both } x \in L \text{ and } x \notin L \text{ have “nifty proofs”}

\text{Is P = NP } \cap \text{ coNP?}

\text{THIS IS AN OPEN QUESTION!}
An Interesting Problem in \( NP \cap \text{coNP} \)

**FACTORIZING**

\[
\{ (m, n) \mid m > n > 1 \text{ are integers, there is a prime factor } p \text{ of } m \text{ where } n \leq p < m \}
\]

**Theorem:** FACTORIZING \( \in \text{NP } \cap \text{coNP} \)

**Theorem:** If FACTORIZING \( \in P \), then there is a polynomial-time algorithm which, given an integer \( n \), outputs either “\( n \text{ is PRIME} \)” or a prime factor of \( n \).
**NP-complete** problems:

SAT, 3SAT, CLIQUE, VC, SUBSET-SUM, ...

**coNP-complete** problems:

UNSAT, TAUTOLOGY, NOHAMPATH, ...

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

Nobody knows if they exist!

P, NP, coNP can be defined in terms of specific machine models, and for every possible machine we can give a simple 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

Polynomial time

Is formula $F$ in SAT?

$q_{\text{YES}}$ yes

yes

ANPUTURE

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 $q_{\text{YES}}$, otherwise $q_{\text{NO}}$)

This notion makes sense even when $M$ runs in *polynomial time* and $B$ is *not* in $P$!
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.
Some Complexity Classes With Oracles

Let B be a language.

\[ P^B = \{ L \mid L \text{ can be decided by some polynomial-time } TM \text{ with an oracle for } B \} \]

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

\[ P^{NP} = \text{the class of languages decidable by some polynomial-time oracle TM with an oracle for some } B \text{ in NP} \]
Is \( P^{\text{SAT}} \subseteq P^{\text{NP}} \)?

Yes! By definition...

Is \( P^{\text{NP}} \subseteq P^{\text{SAT}} \)?

Yes!

Every NP language can be reduced to SAT!

Let \( M^B \) be a poly-time TM with oracle \( B \in \text{NP} \). We define \( N^{\text{SAT}} \) that simulates \( M^B \) step for step. When the sim of \( M^B \) makes query \( w \) to oracle \( B \), \( N^{\text{SAT}} \) reduces \( w \) to a formula \( \phi_w \) in poly-time, then calls its oracle for SAT on \( \phi_w \)
Is $\text{NP} \subseteq \text{P}^{\text{NP}}$?

Yes!

*Just ask the oracle for the answer!*

For every $L \in \text{NP}$ define an oracle TM $M^L$ which asks the oracle if the input is in $L$, then outputs the answer.
Is \( \text{coNP} \subseteq \text{P}^{\text{NP}} \)?

Yes!

Again, just ask the oracle for the answer!

For every \( L \in \text{coNP} \) we have \( \neg L \in \text{NP} \)

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, \( \text{P}^{\text{NP}} = \text{P}^{\text{coNP}} \) and \( \text{P}^{\text{SAT}} = \text{P}^{\text{TAUT}} \)
For every poly-time TM $M$ with oracle $B \in P$, we can simulate each query $z$ to oracle $B$ by simply running a polynomial-time decider for $B$.

Is $P^B \subseteq P$? Yes!

Suppose $B$ is in $P$. The resulting machine runs in polynomial time!
\( P^{\text{NP}} \) = the class of languages decidable by some polynomial-time oracle TM \( M^B \) for some \( B \) in NP

**Informally:** \( P^{\text{NP}} \) is the class of problems you can solve in polynomial time, assuming a SAT solver which gives you answers quickly.
\( \mathsf{P}^{\mathsf{NP}} \) = the class of languages decidable by some polynomial-time oracle TM \( M^B \) for some \( B \) in \( \mathsf{NP} \)

Informally, \( \mathsf{P}^{\mathsf{NP}} \) is the class of problems you can solve in polynomial time, if SAT solvers work.

A problem in \( \mathsf{P}^{\mathsf{NP}} \) that looks harder than SAT or TAUT:

\[
\text{FIRST-SAT} = \{ (\phi, i) \mid \phi \in \text{SAT} \text{ and the } i\text{th bit of the lexicographically first SAT assignment of } \phi \text{ is 1} \}
\]

Using polynomially many calls to SAT, we can compute the lexicographically first satisfying assignment.

“Theorem” FIRST-SAT is \( \mathsf{P}^{\mathsf{NP}} \)-complete.
Is \( \text{NP}^B = \text{NP} \)?

It is believed the answers are NO ...

\[ \text{NP}^B = \{ L \mid L \text{ can be decided by a polynomial-time nondeterministic TM with an oracle for } B \} \]

\[ \text{coNP}^B = \{ L \mid L \text{ can be decided by a poly-time co-nondeterministic TM with an oracle for } B \} \]

Is \( \text{NP} = \text{NP}^{\text{NP}} ? \)

Is \( \text{coNP}^{\text{NP}} = \text{NP}^{\text{NP}} ? \)

**THESE ARE OPEN QUESTIONS!**
Logic Minimization is in $\text{coNP}^{\text{NP}}$

Two Boolean formulas $\phi$ and $\psi$ 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 $x \lor x$ equivalent? \textbf{Yes}
- Are $x$ and $x \lor \neg x$ equivalent? \textbf{No}
- Are $(x \lor \neg y) \land \neg(\neg x \land y)$ and $x \lor \neg y$ equivalent? \textbf{Yes}

A Boolean formula $\phi$ is \textbf{minimal} if no \textit{smaller} formula is equivalent to $\phi$ (count number of $\lor$, $\land$, $\neg$, and variable occurrences).

$$\text{MIN-FORMULA} = \{ \phi \mid \phi \text{ is minimal} \}$$
Theorem: $\text{MIN-FORMULA} \in \text{coNP}^{\text{NP}}$

Proof:

Define $\text{NEQUIV} = \{ (\phi, \psi) \mid \phi \text{ and } \psi \text{ are not equivalent} \}$

Observation: $\text{NEQUIV} \in \text{NP}$ (Why?)

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

Given a formula $\phi$,

Try all formulas $\psi$ such that $\psi$ is smaller than $\phi$.

If $((\phi, \psi) \in \text{NEQUIV})$ then accept else reject

$\text{MIN-FORMULA}$ is not known to be in $\text{coNP}$ or $\text{NP}^{\text{NP}}$
The Difficulty of Formula Minimization

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

Theorem: MIN-CNF-FORMULA is coNP^NP-complete

Proof: Beyond the scope of this course...

Note: We don’t know if MIN-FORMULA is coNP^NP complete!
FACTORING
coNP
TAUT
NP
MIN-FORMULA
P
coNP
PNP
NP
FACTORIZING
SAT
Oracles and P vs NP

*Everything* about TMs we have proved in this class also works for TMs with arbitrary oracles.

**Theorem [Baker, Gill, Solovay ’75]:**

1. There is an oracle B where $P^B = NP^B$
2. There is an oracle A where $P^A \neq NP^A$

See Sipser 9.2

**Moral:** Any proof technique that extends to Turing Machines with arbitrary oracles won’t be able to resolve P versus NP!

THE “RELATIVIZATION BARRIER”
Space Complexity
Measuring Space Complexity

We measure space complexity by finding the largest tape index reached during the computation.
Let M be a deterministic Turing machine (not necessarily halting)

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

**Definition:** \( \text{SPACE}(S(n)) = \{ L \mid L \text{ is decided by a Turing machine with } O(S(n)) \text{ space complexity} \} \)
Theorem: \( 3\text{SAT} \in \text{SPACE}(n) \)

Proof Idea: Try all possible assignments to the (at most \( n \)) variables in a formula \( \phi \) of length \( n \). Accept iff an assignment makes \( \phi \) true. This can be done in \( O(n) \) space.

Theorem: \( \text{NTIME}(t(n)) \) is in \( \text{SPACE}(t(n)) \)

Proof Idea: 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 (store a sequence of \( t(n) \) transitions).
Theorem: Let $s : \mathbb{N} \rightarrow \mathbb{N}$ satisfy $s(n) \geq n$, for all $n$. Then every $s(n)$ space multi-tape TM has an equivalent $O(s(n))$ space one-tape TM.

The simulation of multitape TMs by one-tape TMs already achieves this!

Corollary: The number of tapes doesn’t matter for space complexity! One tape TMs are as good as any other model!
Space Hierarchy Theorem

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

**Theorem:** For functions \( s, S : \mathbb{N} \rightarrow \mathbb{N} \) where \( s(n)/S(n) \rightarrow 0 \)

\[
\text{SPACE}(s(n)) \subsetneq \text{SPACE}(S(n))
\]

**Proof Idea:** Diagonalization

Make a Turing machine \( N \) that on input \( M \), simulates the TM \( M \) on input \( M \) using up to \( S(|M|) \) space, then flips the answer.

Show \( L(N) \) is in \( \text{SPACE}(S(n)) \) but not in \( \text{SPACE}(s(n)) \)
\[ PSPACE = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k) \]

Since for every \( k \), \( \text{NTIME}(n^k) \) is in \( \text{SPACE}(n^k) \), we have:

\[ P \subseteq NP \subseteq PSPACE \]
The class PSPACE formalizes the set 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 $n^2$ time to solve!

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

Is P = PSPACE?