6.045

Lecture 17: Finish NP-Completeness, coNP and Friends

Definition: A language B is **NP-complete** if:

1. $B \in NP$
2. Every A in NP is poly-time reducible to B
   That is, $A \leq_P B$
   When this is true, we say "B is NP-hard"

Last time: We showed

3SAT $\leq_P$ CLIQUE $\leq_P$ IS $\leq_P$ VC $\leq_P$ SUBSET-SUM $\leq_P$ KNAPSACK

All of them are in NP, and 3SAT is NP-complete, so all of these problems are NP-complete!

The Knapsack Problem

Input: S = $\{(v_1,c_1)\ldots, (v_n,c_n)\}$ of pairs of positive integers (items)

a capacity budget C

a value target V

Decide: Is there an $S' \subseteq \{1,\ldots,n\}$ such that $\sum_{i\in S'} v_i \geq V$ and $\sum_{i\in S'} c_i \leq C$?

Define: KNAPSACK = {(S, C, V) | the answer is yes}

A classic economics/logistics/OR problem!

Theorem: KNAPSACK is NP-complete

KNAPSACK is NP-complete

KNAPSACK is in NP?

Theorem: SUBSET-SUM $\leq_P$ KNAPSACK

Proof: Given an instance $(S = \{a_1,\ldots,a_n\}, t)$ of SUBSET-SUM, create a KNAPSACK instance:

For all i, set $(v_i, c_i) := (a_i, a_i)$

Define $T = \{(v_1, c_1),\ldots, (v_n, c_n)\}$

Define $C := V := t$

Then, $(S,t) \in$ SUBSET-SUM $\Leftrightarrow$ $(T,C,V) \in$ KNAPSACK

Subset of S that sums to t =

Solution to the Knapsack instance!

The Partition Problem

Input: Set $S = \{a_1, \ldots, a_n\}$ of positive integers

Decide: Is there an $S' \subseteq S$ where $(\sum_{i \in S'} a_i) = (\sum_{i \in S - S'} a_i)$?

(Formally: PARTITION is the set of all encodings of sets S such that the answer to the question is yes.)

In other words, is there a way to partition S into two parts, so that both parts have equal sum?

A problem in Fair Division:

Think of a_i as "value" of item i. Want to divide a set of items into two parts S' and $S - S'$, of the same total value.
Give S' to one party, and $S - S'$ to the other.

Theorem: PARTITION is NP-complete

PARTITION is NP-complete

(1) PARTITION is in NP

(2) SUBSET-SUM $\leq_P$ PARTITION

Input: Set $S = \{a_1, \ldots, a_n\}$ of positive integers
positive integer t

Reduction: If $t > \sum_i a_i$ then output $\{1,2\}$
Else output $T := \{a_1, \ldots, a_n, 2A-t, A+t\}$, where $A := \sum_i a_i$

Claim: $(S,t) \in$ SUBSET-SUM $\Leftrightarrow T \in$ PARTITION
That is, S has a subset that sums to t
$\Leftrightarrow$ T can be partitioned into two sets with equal sums
Easy case: $t > \sum_i a_i$

Input: Set $S = \{a_1,\ldots, a_n\}$ of positive integers, positive t

Output: $T := \{a_1,\ldots, a_n, 2A-t, A+t\}$, where $A := \sum_i a_i$

Claim: $(S,t) \in$ SUBSET-SUM $\Leftrightarrow T \in$ PARTITION

What's the sum of all numbers in T? **4A**

Therefore: $T \in$ PARTITION

$\Leftrightarrow$ There is a $T' \subseteq T$ that sums to 2A.

Proof of $(S,t) \in$ SUBSET-SUM $\Rightarrow T \in$ PARTITION:

If $(S,t) \in$ SUBSET-SUM, then let $S' \subseteq S$ sum to t.

The set $S' \cup \{2A-t\}$ sums to 2A, so $T \in$ PARTITION

Input: Set $S = \{a_1,\ldots, a_n\}$ of positive integers, positive t

Output: $T := \{a_1,\ldots, a_n, 2A-t, A+t\}$, where $A := \sum_i a_i$

Remember: sum of all numbers in T is 4A.

Claim: $(S,t) \in$ SUBSET-SUM $\Leftrightarrow T \in$ PARTITION

$T \in$ PARTITION $\Leftrightarrow$ There is a $T' \subseteq T$ that sums to 2A.

Proof of: $T \in$ PARTITION $\Rightarrow (S,t) \in$ SUBSET-SUM

If $T \in$ PARTITION, let $T' \subseteq T$ be a subset that sums to 2A.

Observation: Exactly *one* of $\{2A-t, A+t\}$ is in T'.

If $(2A-t) \in T'$, then $T' - \{2A-t\}$ sums to t. By Observation, the set $T' - \{2A-t\}$ is a subset of S. So $(S,t) \in$ SUBSET-SUM.

If $(A+t) \in T'$, then $(T - T') - \{2A-t\}$ sums to $(2A - (2A-t)) = t$

By Observation, $(T - T') - \{2A-t\}$ is a subset of S.

Therefore $(S,t) \in$ SUBSET-SUM in this case as well.

The Bin Packing Problem

Input: Set S = {a_1,…, a_n} of positive integers,
a bin capacity **B**, and a number of bins **K**.

Decide: Can S be partitioned into disjoint subsets $S_1, \ldots, S_k$ such that each S_i sums to at most B?

Think of a_i as the capacity of item i.
Is there a way to pack the items of S into K bins,
where each bin has capacity B?

Ubiquitous problem in shipping and optimization!

Theorem: BIN PACKING is NP-complete

BIN PACKING is NP-complete

(1) BIN PACKING is in NP (Why?)

(2) PARTITION $\leq_P$ BIN PACKING

Proof: Given an instance $S = \{a_1, \ldots, a_n\}$ of PARTITION, output an instance of BIN PACKING with:

$$S = \{a_1, \ldots, a_n\}$$
$$B = (\textstyle\sum_i a_i)/2$$
$$k = 2$$

Then, $S \in$ PARTITION $\Leftrightarrow$ $(S,B,k) \in$ BIN PACKING:

There is a partition of S into two equal sums iff there is a solution to this Bin Packing instance!

Two Problems

Let G denote a graph, and s and t denote nodes.

SHORTEST PATH
= {(G, s, t, k) |
G has a simple path of < k edges from s to t }

LONGEST PATH
= {(G, s, t, k) |
G has a simple path of $\geq$ k edges from s to t }

Are either of these in P? Are both of them?

HAMPATH = { (G,s,t) | G is an directed graph with a Hamiltonian path from s to t}

Theorem: HAMPATH is NP-Complete

(1) HAMPATH $\in$ NP

(2) 3SAT $\leq_P$ HAMPATH

Sipser (p.314-318) and recitation!

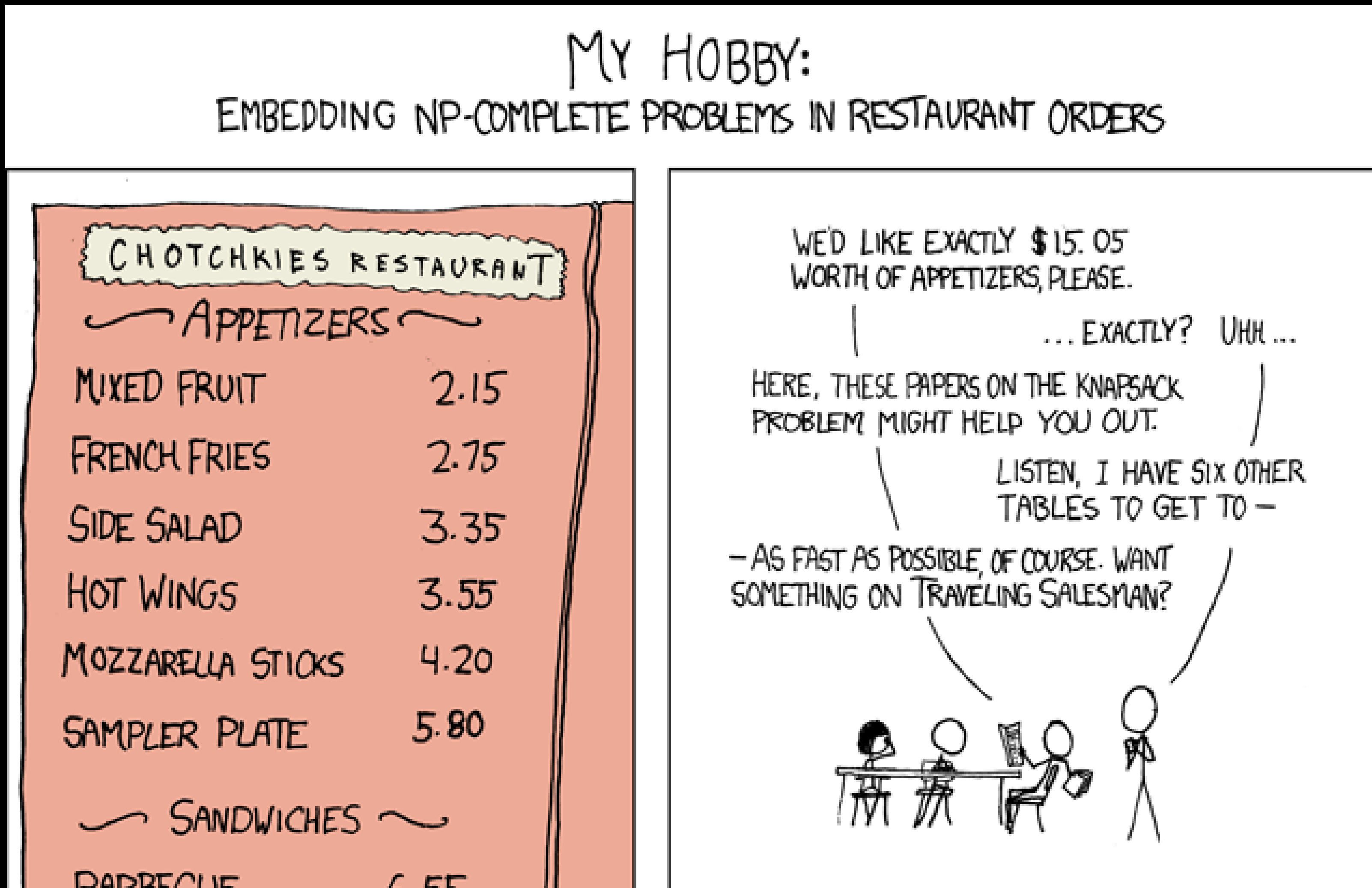
HAMPATH $\leq_P$ LONGEST-PATH

LONGEST-PATH
= {(G, s, t, k) |
G has a simple path of $\geq$ k edges from s to t }

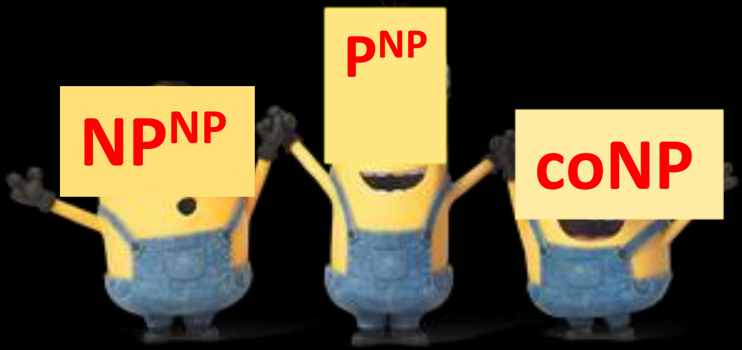
Can reduce HAMPATH to LONGEST-PATH
by observing:

(G, s, t) $\in$ HAMPATH
$\Leftrightarrow$ (G, s, t, $|V|$-1) $\in$ LONGEST-PATH

Therefore LONGEST-PATH is NP-hard.


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO —
— AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

coNP and Friends



(Note: any resemblance to other characters, living or animated, is purely coincidental)

NP: "Nifty Proofs"

For every L in NP,

if $x \in L$ then there is a "short proof" that $x \in L$:

$L = \{x \mid \exists y$ of poly($|x|$) length so that $V(x,y)$ accepts$\}$

But if $x \notin L$, there might not be a short proof!

There is an asymmetry between
the strings in L and strings not in L.

Compare with a *recognizable* language L:

Can always verify $x \in L$ in *finite time* (a TM accepts x),
but if $x \notin L$, that could be because
the TM goes in an *infinite loop* on x!

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

What does a coNP problem L look like?

The instances NOT in L have *nifty proofs*.

Recall we can write any NP problem L in the form:

L = {x | $\exists$y of poly(|x|) length so that V(x,y) accepts}

Therefore:

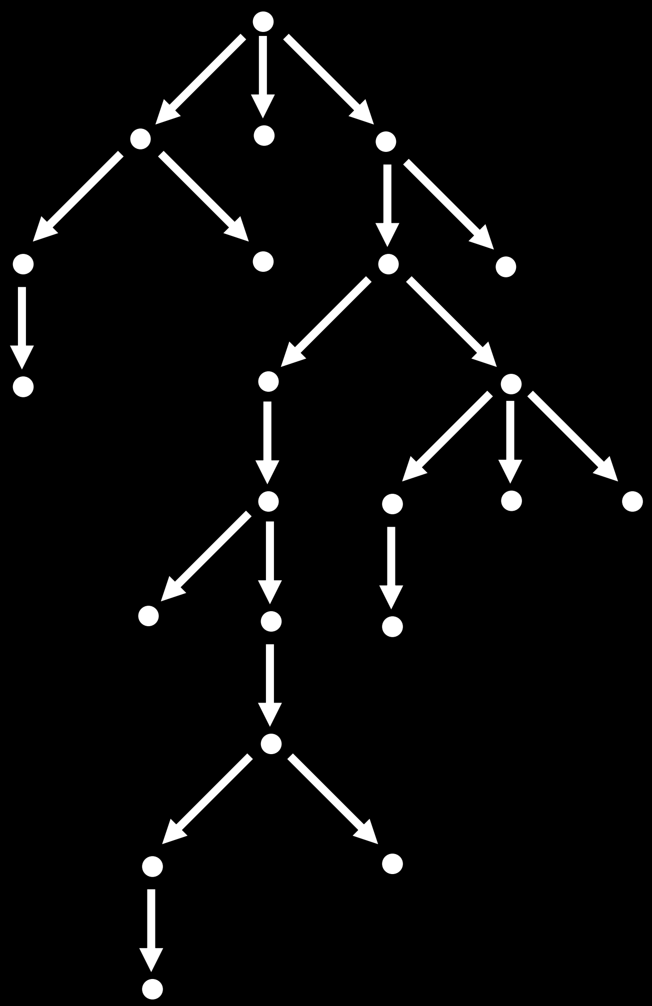
$\neg$L = {x | $\neg\exists$y of poly(|x|) length so that V(x,y) accepts}

= {x | $\forall$y of poly(|x|) length, V(x,y) rejects}

Instead of using an "existentially guessing" (nondeterministic) machine,
we can define a "universally verifying" machine!

Definition: coNP = { L | ¬L ∈ NP }

What does a coNP computation look like?

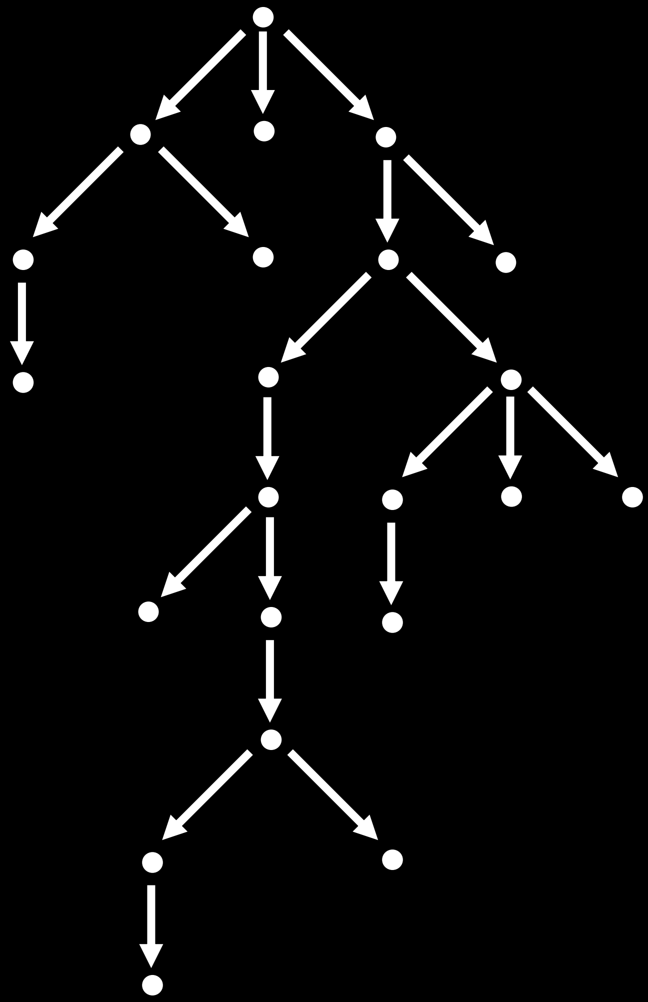


A ***co-nondeterministic*** **machine** has multiple computation paths, and has the following behavior:

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

Definition: coNP = { L | ¬L ∈ NP }

What does a coNP computation look like?



In NP algorithms, we can use a "guess" instruction in pseudocode:
Guess string y of $k|x|^k$ 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 $k|x|^k$ 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?

Is P $\subseteq$ coNP?

Yes!

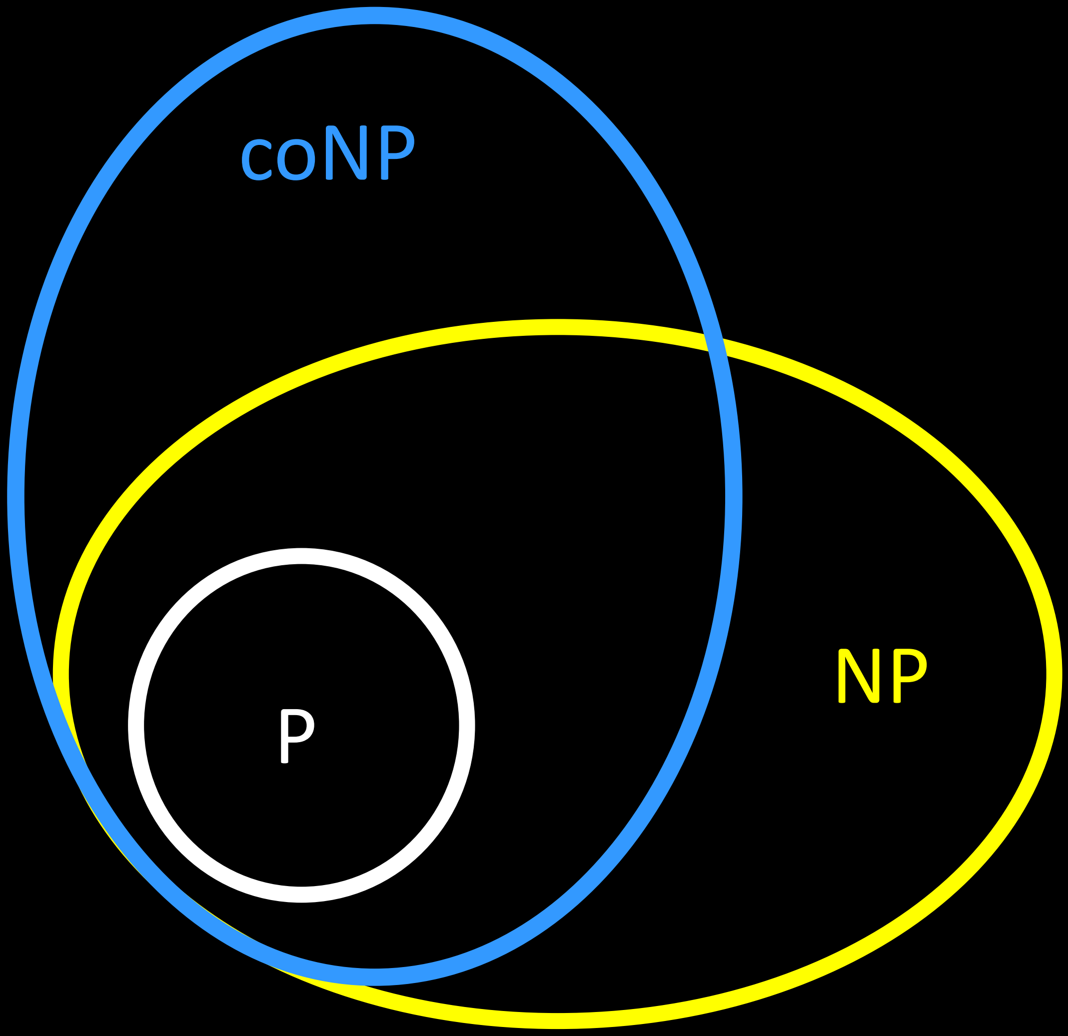
L $\in$ P implies that $\neg$L $\in$ P
(hence $\neg$L $\in$ NP)

In general, *deterministic* complexity classes are closed under complement

Is NP = coNP?

THIS IS AN OPEN QUESTION!

It is believed that NP $\neq$ coNP


coNP
NP
P

coNP = { L | ¬L ∈ NP }

Definition: A language B is coNP-complete if

1. B ∈ coNP

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

Key Trick: Can use A $\leq_P$ B ⇔ ¬A $\leq_P$ ¬B to turn NP-hardness into co-NP hardness

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

Theorem: UNSAT is coNP-complete

Proof: (1) UNSAT $\in$ coNP   *(why?)*

(2) UNSAT is coNP-hard:

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

Since $\neg A \in NP$, we have $\neg A \leq_P$ 3SAT by the Cook-Levin theorem. This reduction already works!

$w \in \neg A \Rightarrow \phi_w \in 3SAT$

$w \notin A \Rightarrow \phi_w \notin UNSAT$

$w \notin \neg A \Rightarrow \phi_w \notin 3SAT$

$w \in A \Rightarrow \phi_w \in UNSAT$

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 }
= $\{\phi \mid \neg\phi \in$ UNSAT$\}$

Theorem: TAUTOLOGY is coNP-complete

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

(2) TAUTOLOGY is coNP-hard:

UNSAT $\leq_P$ TAUTOLOGY:
Given Boolean formula ϕ, output $\neg\phi$

NP ∩ coNP = { L | L and ¬L ∈ NP }

L ∈ NP ∩ coNP means that
both $x \in$ L *and* $x \notin$ L have "nifty proofs"

Is P = NP ∩ coNP?

THIS IS AN OPEN QUESTION!

Is P = NP ∩ coNP?

Why might this be true?

Analogy with computability

Why might this be false?

If it's true, most crypto fails!

An Interesting Problem in NP ∩ coNP

FACTORING
= { (n, k) | n > k > 1 are integers written in binary,
and there is a prime factor p of n where k ≤ p < n }

If FACTORING ∈ P, we could potentially use the algorithm to factor every integer, and break RSA!
Can binary search on k to find a prime factor of n.
More details in slides posted online

Theorem: FACTORING ∈ NP ∩ coNP

PRIMES = {n | n is a prime number written in binary}

Theorem (Pratt '70s): **PRIMES $\in$ NP $\cap$ coNP**

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
= { (n, k) | n > k > 1 are integers written in binary,
there is a prime factor p of n where $k \le p < n$ }

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

Proof: (1) FACTORING $\in$ NP

A prime factor p of n such that $p \ge k$ is a proof that (n, k) is in FACTORING
(can check primality in P*, can check p divides n in* P*)*

(2) FACTORING $\in$ coNP

The prime factorization $p_1^{e1} \ldots p_m^{em}$ of n is a proof that (n, k) is not in FACTORING:

Verify each p_i is prime in P, and that $p_1^{e1} \ldots p_m^{em} = n$

Verify that for all i=1,…,m that $p_i < k$

FACTORING
= { (n, k) | n > k > 1 are integers written in binary,
there is a prime factor p of n where $k \le p < n$ }

Theorem: If FACTORING $\in$ P, then there is a polynomial-time algorithm which, given an integer n, outputs either "n is PRIME" or a prime factor of n.

Idea: Binary search for the prime factor!

Given binary integer n, initialize an interval [2,n].

If (n, 2) is not in FACTORING then output "PRIME"

If $(n, \lceil n/2 \rceil)$ is in FACTORING then

shrink interval to $[\lceil n/2 \rceil, n]$ (set $k := \lceil 3n/4 \rceil$)

else, shrink interval to $[2, \lceil n/2 \rceil]$ (set $k := \lceil n/4 \rceil$)

Keep picking k to halve the interval after each (n,k) call to FACTORING. Takes O(log n) calls to FACTORING!