CSE396 Lecture Tue. 5/4: Computation Checking and Complexity

Reductions Via Computation Traces

We recall (from the April 8 lecture) the definition of **instantaneous descriptions** (**IDs**, also called **configurations**), which give the current state, current tape contents aside from blanks, and current head position(s) at any point in a computation by a Turing machine. The starting ID on an input $x \in \Sigma^*$ is denoted by $I_0(x)$. For a single-tape Turing machine M with start state s this can have the simple form $I_0(x) = sx$ where the state is treated as a character. If we make TMs do "good housekeeping" when they are about to produce an output y by blanking out everything on their tape(s) except for y, then the computations can end in a unique final ID $I_f = q_{acc}y$. If the TM is a decider, we can suppose it outputs 1 for "accept" and 0 for "reject". Then it has a unique accepting ID $I_f = q_{acc}1$. We also defined the relation $I \vdash_M J$ to mean the ID I can go to the ID J in a single step by M. Thus a **valid accepting computation** (**trace**) has the form

$$I_0(x) \vdash_M I_1 \vdash_M I_2 \vdash_M I_3 \vdash_M \cdots \vdash_M I_{t-2} \vdash_M I_{t-1} \vdash_M I_f\,,$$

and a valid computation that halts and rejects can be defined analogously with $I_{rej} = q_{rej}0$ as the last ID I_t. Then t is the **number of steps**---that is, the **time** taken by the computation---and we generally suppose this is at least $n+1$ where n is the length of x. The computation trace itself can be encoded as a string

$$\vec{c} = \langle I_0(x), I_1, I_2, I_3, \ldots, I_{t-2}, I_{t-1}, I_t \rangle\,.$$

of length $O(t^2)$, since the IDs can expand by at most one char in each step. The key question is:

> What kinds of machines---or combinations Z of machines or other formal objects---can tell whether strings of this kind really represent valid computations?

That is, given any Turing machine M, what does it take to recognize the language V_M of its valid computation traces? Let's write this as a definition and observe a key set of facts:

Definition: For any Turing machine M (wlog. a single-tape deterministic TM), V_M is the language of its valid (accepting) computation traces.

Theorem: $L(M) = \emptyset \iff V_M = \emptyset$. ☒

Note that even if M is not total---indeed even if $L(M)$ is c.e. but undecidable so that M cannot be total---the language V_M can be decidable. This is because you are not just given x but an entire string $w = \langle I_0(x), I_1, I_2, I_3, \ldots, I_{t-2}, I_{t-1}, I_t \rangle$, and you just need to determine by looking entirely within the bounds of w itself whether it is valid. This means checking that

$$I_{k-1} \vdash_M I_k$$

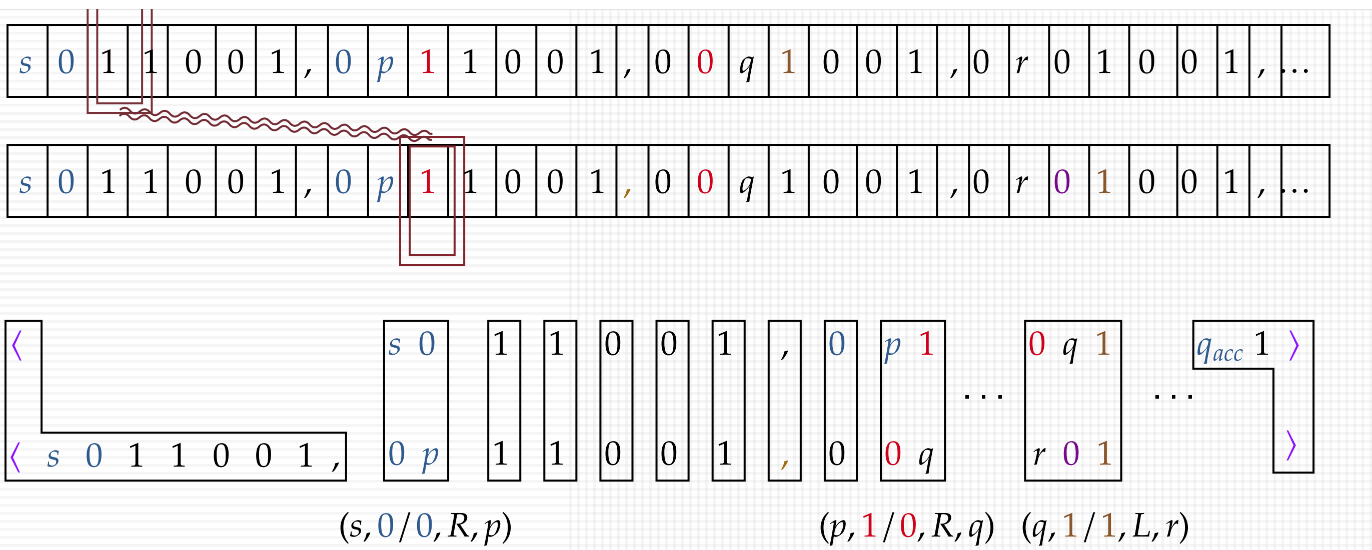
for all k, $1 \le k \le t$. This relation is decidable by checking that the action of some instruction $(q, c/d, D, r)$ in the code of M that is applicable in I_{k-1} (for instance on a single-tape TM, the ID could be $uqcv$ for some strings $u, v \in \Gamma^*$ and $c \in \Gamma$) produces the ID I_k. For example, suppose $x = 011001$ and the first three instructions executed by a single-tape TM M are $(s, 0/0, R, p)$, $(p, 1/0, R, q)$, and $(q, 1/1, L, r)$. Then

$$\vec{c} = \langle s011001, 0p11001, 00q1001, 0r01001, \ldots \rangle$$

The text defines **linear bounded automata** (**LBA**s) as machines to do the check, but I instead like to picture a two-tape kind of DFA, one that "Is-A" deterministic LBA anyway:

Definition (not in the text): A **two-head DFA** (**2HDFA**) is a deterministic two-tape TM that gets its input x initially on *both* tapes, and whose heads may not move left.

A 2HDFA can accept the nonregular language $\{a^n b^n : n \ge 0\}$ by having one head advance to the b^n part (or if the input x is ϵ, accept right away) while the other stays put, and then check a^n against b^n. It can recognize the marked double-word language $DW = \{w\#w : w \in \{0,1\}^*\}$ in a similar manner. And that's essentially why 2HDFAs can check computations:



A two-head DFA checking computations, plus the idea of the Post Correspondence Problem

Note that *most* of the check is that the parts of the IDs away from the "state" part match char-by-char, as in DW. Recall that DW is not a CFL but it is the *complement* of a CFL. Now V_M is like an iterated version of DW. The *complement* of V_M, however, comes down to much the same as the complement of DW. Basically, a string $w = \langle I_0(x), I_1, I_2, \ldots, I_{t-1}, I_t \rangle$ belongs to $\widetilde{V}_M$ if and only if either:

- it doesn't have the correct form as a sequence of IDs, or
- there is a screwup $I_{k-1} \nvdash I_k$ for *some* k: no legal instruction can execute the change, or some other character mismatch.

The first kind of fault can always be detected on the fly---that's another reason we can often ignore the issue of "invalid codes" and assume a given string w has the right "angle-bracket" format. The main point is that if the second happens, it is enough that it happens for *just one k*. Hence a nondeterministic PDA N can guess which k and then *verify* that there is a screwup. (If a branch of N guesses the wrong j, some other branch will guess the right k and accept; or if there is no screwup or other fault, all branches will correctly reject.) The ability of a PDA to detect a mismatch is related to the reason the *complement* of the double-word language *is* a CFL. Thus we conclude:

Lemma: For any Turing machine M, $\widetilde{V}_M$ is a CFL. Moreover, there is a computable mapping h such that $h(\langle M\rangle) = \langle G\rangle$ giving a CFG G such that $L(G) = \widetilde{V}_M$. ☒

This finally brings us to the proof of a long-promised fact:

Theorem: The ALL_{CFG} problem is undecidable.

Proof: $\langle M\rangle \in E_{TM} \equiv L(M) = \varnothing \iff V_M = \varnothing \iff \widetilde{L(G)} = \varnothing \iff L(G) = \Sigma^*$, where G is given by the computable mapping $h(\langle M\rangle)$. ☒

In fact, this is part of a "Meta-Theorem":

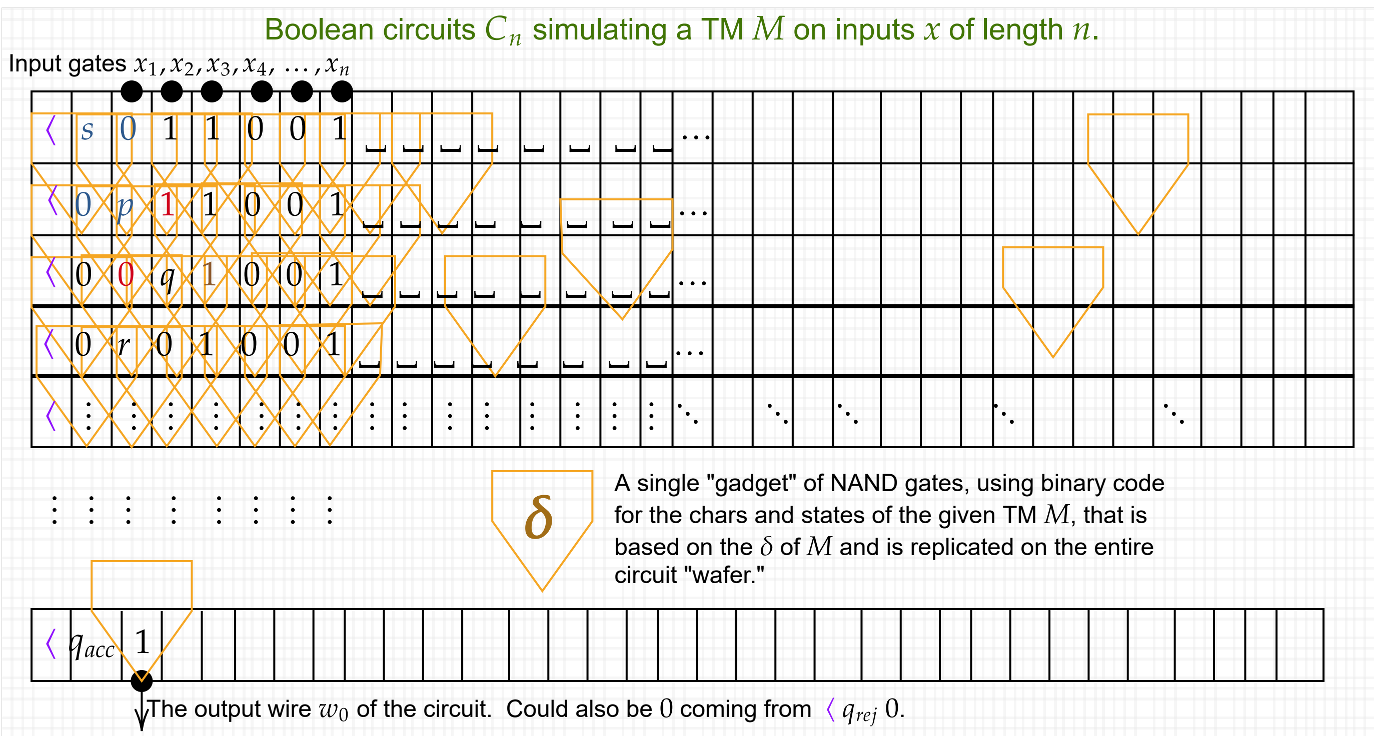
General Theorem: For any type of machine or "machine combo" Z that can verify computation traces, the E_Z problem ("emptiness problem for Z-machines") is undecidable. If the combo represents "broken traces" instead, then ALL_Z is undecidable. ☒

So what other "Z" besides a combo of two (D)PDA/grammars or "complement of a grammar" can do computation-checking? Here is a summary of examples---of which we care most about 5:

1. The complement of the language of a CFG, so ALL_{CFG} is undecidable.
2. A two-head DFA. Since 2HDFAs are deterministic total TMs that can be complemented, both E_{2HDFA} and ALL_{2HDFA} are undecidable.
3. A **Post System**, which (FYI) is defined as a set of **tiles**, each of which has a "top" string and a "bottom" string. You can use as many copies of each tile as desired except for a unique **starting tile**, which can be as shown in the picture with top string "$\langle$" (or just ϵ). Some tiles have shorter bottom string than top string---and inutitively they involve a machine blanking out a character, which it can do at each step in the "good housekeeping" routine mentioned above before accepting---which can involve a **final tile** having "$q_{acc}1\rangle$" is its top string and "$\rangle$" (or ϵ) as

its bottom string. The *goal* is to add tiles after the start tile so that the whole top and bottom strings become equal. We can convert any TM M and input w into a set $T_{M,w}$ of tiles, with "$\langle sw$" as the bottom string of the start tile, that can be solved if and only if M accepts w (so we can solve Post's problem by completing the computation trace). So $\langle M, w \rangle \in A_{TM} \iff T_{M,w}$ is a solvable case of Post's Problem, and so Post's Problem is undecidable. Emil Post published the problem in 1946, but he had related ideas going back to the 1920s.

4. A **linear bounded automaton** (**LBA**), defined as a Turing machine that on any input x uses only the cells initially occupied by x (plus optionally the blanks to the left and right of x, or we can initialize with endmarkers $\wedge\, x\, \$$ or $\langle\ x\ \rangle$ instead). Called **DLBA** when deterministic, else **NLBA**. A 2HDFA "Is-A" DLBA, and DLBAs are closed under complementation, so we've already proved the text's theorems about E_{DLBA} and ALL_{DLBA} being undecidable.
5. **Boolean Circuits**---wlog. of NAND gates only since NAND is a universal gate. They can verify computations by arranging the alleged IDs in a $(t+1) \times (t+1)$ grid, since the space s used by an ID cannot grow to be more than the time t elapsed.



The concept of V_M works also when M is nondeterministic. The circuits C_n can still verify that a given computation branch of the NTM is legal, if the branch is written over the whole "circuit board." But only when M is deterministic can C_n be given just "$\langle I_0(x)$" (followed by the binary code for the rest of the top row being blanks) and then *execute* the rest of the computation, with the 0/1 (no/yes) answer coming on the output wire w_0 as shown. This says that (with only an $O(t^2)$ loss in efficiency that can be cleverly reduced to $O(t \log t)$) **software can be burned into hardware**. We will use this idea when the input includes both x and a potential "witness string" y.

Computational Complexity (Ch. 7 now)

We have talked about the running times of Turing machines and algorithms in general, already. It is finally time to formalize this.

Definition:

1. Given a function $t: \mathbb{N} \to \mathbb{N}$, a DTM M **runs in time** $t(n)$ if for all n and inputs x of length n, $M(x) \downarrow$ within $t(n)$ steps.
2. Given a function $s: \mathbb{N} \to \mathbb{N}$, a DTM M **runs in space** $s(n)$ if for all n and inputs x of length n, $M(x) \downarrow$ while **changing** the character in at most $s(n)$ tape cells.
3. A nondeterministic Turing machine runs within a given time or space bound if **all** of its possible computations obey the bound.

Note that although a computation can "loop" within a finite amount of space, the machine is not regarded as running within that space (in practice, the activation stack or some other tracker would overflow). When the input tape is read-only, the space measure is essentially equivalent to the number of cells accessed on the initially-blank worktapes. For some examples:

- Every DFA runs in time $n+1$. The $+1$ allows an extra step for the Turing machine version of the DFA to go to q_{acc} or q_{rej} on the blank after the input x is all read, depending on whether the original DFA state is accepting or rejecting.
- An NFA might not run in time $n+1$ if its computation uses ϵ-arcs a lot. But it can be efficiently converted into an equivalent NFA without ϵ-arcs, and of course (but not always efficiently) into a DFA, both of which do run in time $n+1$. They all run in *zero* space.
- A 2-head DFA runs in time at most $2n+1$; in worst case, one head advances while the other stays put, then the other catches up. Thus for any DTM or NTM M, the language V_M belongs to $\mathsf{DTIME}[O(n)]$ and hence to P, as well as to DLBA as the text implies.
- A PDA runs in space equal to the maximum size of its stack during a computation, which is most often linear space. It can be made to run in linear time, but the proof is not easy.
- All the problems in Chapter 4, section 4.1, can be decided by Turing machines that run in polynomial time, except for the ones that used converting an NFA or regexp into a DFA.

Definition: For any time function $t(n)$ and space function $s(n)$, using M to mean DTM and $N \ for \ NTM$:

1. $\mathsf{DTIME}[t(n)] = \{L(M): M \ runs \ in \ time \ t(n)\}$
2. $\mathsf{NTIME}[t(n)] = \{L(N): N \ runs \ in \ time \ t(n)\}$
3. $\mathsf{DSPACE}[s(n)] = \{L(M): M \ runs \ in \ space \ s(n)\}$
4. $\mathsf{NSPACE}[s(n)] = \{L(N): N \ runs \ in \ space \ s(n)\}$

Convention: For any collection T of time or space bounds, in particular one defined by O-notation, $\mathsf{DTIME}[T]$ means the union of $\mathsf{DTIME}[t(n)]$ over all functions $t(n)$ in T, and so on.

Definition (some of the "Canonical Complexity Classes"):

1. $\mathsf{P} = \mathsf{DTIME}\left[n^{O(1)}\right]$
2. $\mathsf{NP} = \mathsf{NTIME}\left[n^{O(1)}\right]$
3. $\mathsf{DLBA} = \mathsf{DSPACE}[O(n)] = \{L(M): M \textit{ is a DLBA}\}$.
4. $\mathsf{NLBA} = \mathsf{NSPACE}[O(n)] = \{L(N): N \textit{ is an NLBA}\}$.
5. $\mathsf{PSPACE} = \mathsf{DSPACE}\left[n^{O(1)}\right]$
6. $\mathsf{EXP} = \mathsf{DTIME}\left[2^{n^{O(1)}}\right]$.

The latter equality in lines 3 and 4 is actually a theorem but is pretty immediate. The only two class we know to contain languages not in P is the last one: we know $\mathsf{P} \subsetneq \mathsf{EXP}$. Regarding line 5, it seems we've skipped an analogously-defined class "NPSPACE" but it actually equals PSPACE. Right now P and NP, along with $\text{co-NP} = \{\sim L : L \in \mathsf{NP}\}$ will take center stage, beginning with an important analogy to REC, RE, and co-RE.

Details about NP

The text gives the *verifier* definition of NP. A **verifier** V for a language L decides a predicate $R(x, y)$. Whenever $R(x, y)$ holds we must have $x \in L$, and then y is called a **witness** (or **certificate**) for x being in L. This defines L to belong to NP if:

- V runs in polynomial time (that is, the language $\{\langle x, y\rangle : R(x, y) \textit{ holds}\}$ belongs to P), **and**
- the length of y is bounded by a polynomial p in the length of x.

The theorem that this definition of NP is equivalent to our first one says something even more general about the relation to existential logic and the way RE relates to REC.

Theorem For any language L,

- L is c.e. if and only if there is a polynomial-time decidable predicate $R(x, y)$ such that for all $x \in \Sigma^*$,
$$x \in L \iff (\exists y \in \Sigma^*) R(x, y).$$
- $L \in \mathsf{NP}$ if and only if there are a polynomial-time decidable predicate $R(x, y)$ and a polynomial $p(n)$ such that for all $x \in \Sigma^*$,
$$x \in L \iff (\exists y \in \Sigma^* : |y| \leq p(|x|)) R(x, y).$$

Proof: If we have an NTM N such that $L(N) = L$, then given any x, take y to stand for the code of an accepting computation trace (if any, that is, if $x \in L$):

$$y = \langle I_0(x), I_1, \ldots, I_t\rangle$$

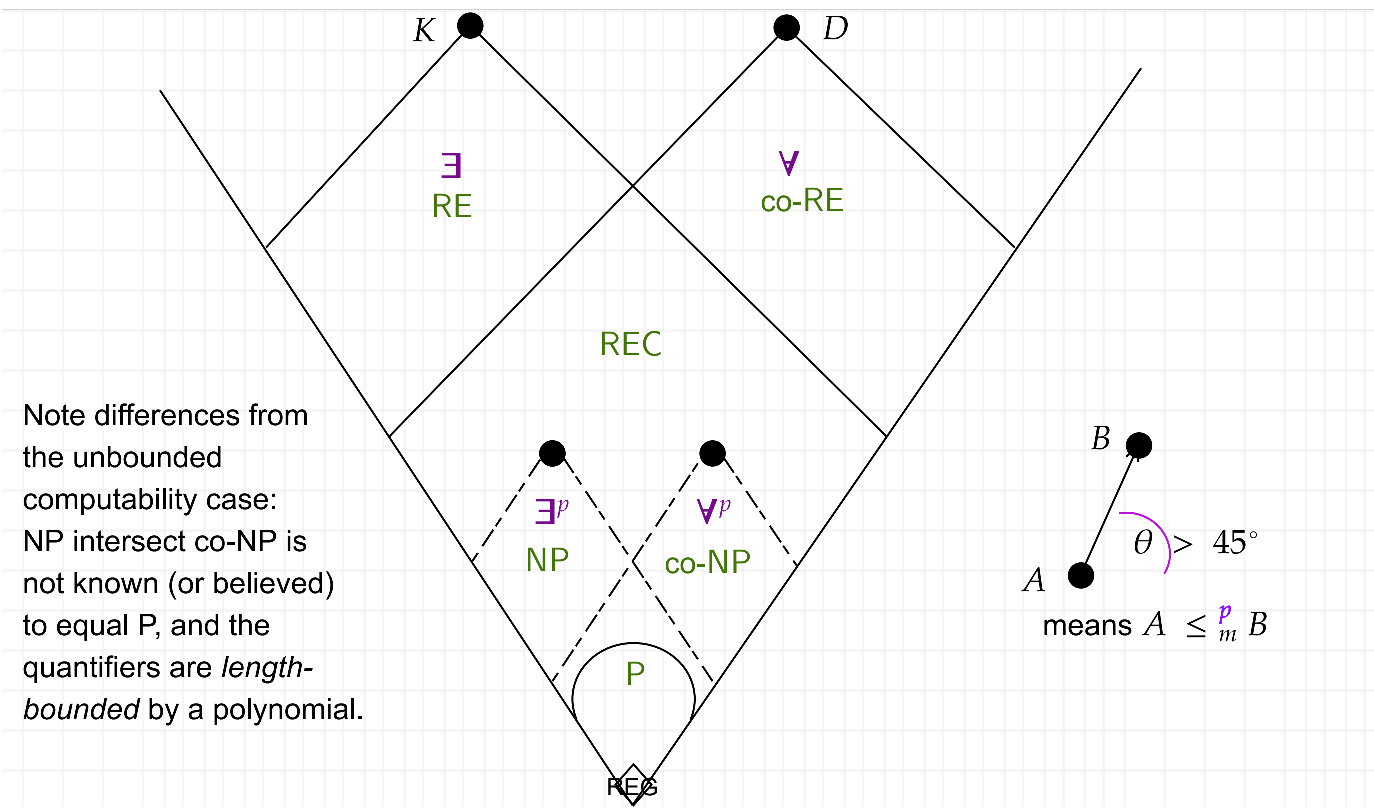
That is, the verifier just decides membership in the language V_N. If N runs in polynomial time $q(n)$, where $n = |x|$, this means $t \leq q(n)$ and so $|y| \leq p(n)$ where $p(n) = O\left(q(n)^2\right)$. And V_N is not only decidable but decidable in time linear in $|y|$, which is likewise polynomial in n.

Going the other way, given a verifier V deciding $R(x, y)$ per above, we can build an NTM N that on any input x uses nondeterministic steps to "guess" a string y and then runs V on $\langle x, y\rangle$. The branch of N accepts x if V accepts $\langle x, y\rangle$, and per above, some such y exists if and only if $x \in L$. Thus $L(N) = L$, so L is c.e.---and if $|y|$ is at most a polynomial in $|x|$, then N runs in polynomial time, which puts such an L into NP. ☒

Corollary: For any language L',

- L' is co-c.e. if and only if there is a polynomial-time decidable predicate $R'(x, y)$ such that for all $x \in \Sigma^*$,
$$x \in L' \iff \left(\forall y \in \Sigma^*\right) R'(x, y).$$
- $L' \in$ **co-NP** if and only if there are a polynomial-time decidable predicate $R'(x, y)$ and a polynomial $p(n)$ such that for all $x \in \Sigma^*$,
$$x \in L' \iff \left(\forall y \in \Sigma^* : |y| \leq p(|x|)\right) R'(x, y).$$ ☒

This yields the analogy that furnishes the gut-check reason for believing NP $\neq$ P and NP $\neq$ co-NP the way we showed RE $\neq$ REC and RE $\neq$ co-RE:



Problems in NP and co-NP

[Lecture got as far as SAT, 3SAT, G3C,and TAUT]

It is usually easiest to tell that (the language of) a decision problem belongs to NP by thinking of a witness and its verification. For example:

Satisfiability (SAT):
Instance: A logical formula ϕ in variables $x_1, \ldots, x_n$ and operators $\wedge, \vee, \neg$.
Question: Does there exist a truth assignment $a \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

The assignment cannot have length longer than the formula, and *evaluating* a formula on a given assignment is quick to do. Hunting for a possible *satisfying assignment*, on the other hand, takes up to 2^n tries if there is no better way than brute force. This is apparently hard even when the Boolean formula has a simple form.

Definition. A Boolean formula is in **conjunctive normal form** (CNF) if it is a conjunction of **clauses**

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where each clause C_j is a disjunction of **literals** x_i or $\bar{x}_i$. The formula is in k**-CNF** if each clause has at most k distinct literals (*strictly* so if each has exactly k).

3SAT
Instance: A Boolean formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in 3CNF.
Question: Is there an assignment $\vec{a} = a_1 a_2 \cdots a_n \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

Now for a problem with a different kind of witness:

Graph Three-Coloring (G3C):
Instance: An undirected graph $G = (V, E)$.
Question: Does there exist a 3-coloring of the nodes of G?

A *3-coloring* is a function $\chi: V \to \{R, G, B\}$ such that for all edges $(u, v) \in E$, $\chi(u) \neq \chi(v)$. The table for χ needs only n entries where $n = |V| \ll N = |G|$, so it has length at most linear in the encoding length N of G (often $N \approx n^2$). And it is easy to *verify* that a *given* coloring χ is correct.

PRIMES $= \{2,3,5,7,11,13,17,19,23, \ldots\}$ (encoded as, say, $10, 11, 101, 111, 1011, \ldots$)

This language was formally shown to belong to P only in 2004, but had long been known to be "almost there" in numerous senses. But now consider this one:

FACT:
Instance: An integer N and an integer k.
Question: Does N have a prime factor p such that $p \leq k$?

If you can always answer yes/no in polynomial time $r(n)$, where $n \approx \log_2 N$ is the number of bits in N, then you can do *binary search* to *find* a factor p of N in time $O(nr(n))$. By doing $N' = n/p$ and repeating you can get the complete factorization of N in polynomial time. This is something that the human race currently does **not** want us to be able to solve efficiently, as it would (more than Covid?) "destroy the world economy" by shredding the basket in which most of our security eggs are still placed. (This is the gist of the 1992 movie *Sneakers* with Robert Redford heading an all-star cast.) But to indicate proximity to this peril, we note:

FACT: **FACT** is in NP $\cap$ co-NP.

Proof: The witness for "no" as well as "yes" is the unique prime factorization $N =: p_1^{a1} p_2^{a2} \cdots p_\ell^{a\ell}$. Although the right-hand side may seem long, ℓ cannot be bigger than the number of bits of N in binary because each p_i is at least 2, and bigger powers only make ℓ have to be smaller. The length of the factorization is $O(n)$. To verify it, one must verify that each p_i is prime---but this is in polynomial time as above---and then simply multiply everything together and check that the result is N. Finally, to verify the *yes* answer, check that at least one of the p_i is $\leq k$; *no* if none.

TAUT:
Instance: A Boolean formula ϕ', same as for SAT.
Question: Is ϕ' a **tautology**, that is, true for all assignments?

Note that ϕ is unsatisfiable $\equiv$ every assignment a makes $\phi(a)$ false $\iff$ every assignment a makes $\phi'(a)$ true, where $\phi' = \neg\phi$. Thus TAUT is essentially the complement of SAT.