CSE491/596 Lecture Wed. 10/21: Computational Complexity Theory

Computational complexity theory is the study of the time, space, and other computational resources needed to solve specified problems, and of the mathematical relationships between problems. The resources are measured on model(s) of computation, of which the deterministic multitape Turing machine (with read-only input tape) is (IMHO) surprisingly realistic. The relationships include reductions. We have already talked about running in linear and polynomial time, but here is the full formalization:

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. When the machine has an output tape, one must specify whether its cells count as used space (usually not).

Some Simplifying Assumptions:

- We can allow real-valued functions such as $t(n) = n \log n$ with the understanding that $t(n)$ really means $\lceil t(n) \rceil$.
- We presume that every TM reads all of its input x plus the blank to its right, so that its running time is at least $n + 1$ (where $n = |x|$ is understood). This can be considered as included under the definition of "good housekeeping." (Random-access machines can meaningfully run in $o(n)$ time, even $O(\log n)$ time, but we will mostly avoid them.)
- Except for machines that run in constant space (which are equivalent to DFAs), the smallest space functions we will consider are $\Omega(\log n)$.
- Unless otherwise specified, we will consider only time and space bounds built out of the mathematical operators $+, -, *, /, \exp, \log$. Note that powers and roots are included, e.g., $\sqrt{z} = \exp(0.5 \log z)$.

The last assumption removes the need to define notions of [*full*] *time and space constructibility*, though we will point out where this feature is being used in proofs. The mathematicians G.H. Hardy and J.E. Littlewood proved a second useful **trichotomy** feature of these functions:

Theorem: If f and g are real functions built up out of $+, -, *, /, \exp, \log$ (but especially not $\sin$ or $\cos$ or other trig functions), then either $f = o(g)$, $f = \Theta(g)$, or $f = \omega(g)$ [i.e., $g = o(f)$]. ☒

We can call these "*H-L functions*" after Hardy and Littlewood; there is a related notion called *elementary functions* but those sometimes allow trig functions. It is not clear to me whether the proof that every H-L function bigger than $n+1$ is fully time constructible (likewise every H-L function that is $\Omega(\log n)$ is fully space constructible) has ever been written down by any human being, but we all use it... From now on, the terms *time function* $t(n)$ and *space function* $s(n)$ include all these assumptions.

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

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

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. $\mathsf{L} = \mathsf{DSPACE}[O(\log n)]$ (Also called DLOG)
4. $\mathsf{NL} = \mathsf{NSPACE}[O(\log n)]$ (Also called NLOG)
5. $\mathsf{PSPACE} = \mathsf{DSPACE}\left[n^{O(1)}\right]$
6. $\mathsf{E} = \mathsf{DTIME}\left[2^{O(n)}\right]$
7. $\mathsf{EXP} = \mathsf{DTIME}\left[2^{n^{O(1)}}\right]$.

We will see that the analogously-defined class "$\mathsf{NPSPACE}$" actually equals PSPACE. No such identity is known for the analogously-defined classes NE and NEXP with regrad to their deterministic counterparts, but we will not be so much concerned with them. There is a sense in which E has a "second-class status" along with $\mathsf{DLIN} = \mathsf{DTIME}[O(n)]$, $\mathsf{NLIN} = \mathsf{NTIME}[O(n)]$, $\mathsf{DLBA} = \mathsf{DSPACE}[O(n)]$, and $\mathsf{NLBA} = \mathsf{NSPACE}[O(n)]$, in that these classes are not closed downward under the applicable reducibility relations, but...that is getting ahead of our story. Right now P and NP, along with $\mathsf{co\text{-}NP} = \{\sim L : L \in \mathsf{NP}\}$ will take center stage, beginning with an important analogy to REC, RE, and $\mathsf{co\text{-}RE}$. When we say that a predicate $R(x,y)$ is polynomial-time decidable (etc.), we mean that the language $\{\langle x,y\rangle : R(x,y) \text{ holds}\}$ belongs to P (etc.)

Theorem (connecting Theorems 10.2 and 13.12 in Debray's notes): For any language L,

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

In both cases, $R(x, y)$ can be linear-time decidable; indeed, R can be the Kleene T-predicate $T(N, x, \vec{c})$ as defined also for NTMs. (The key difference is the polynomial length bound q, not R.)

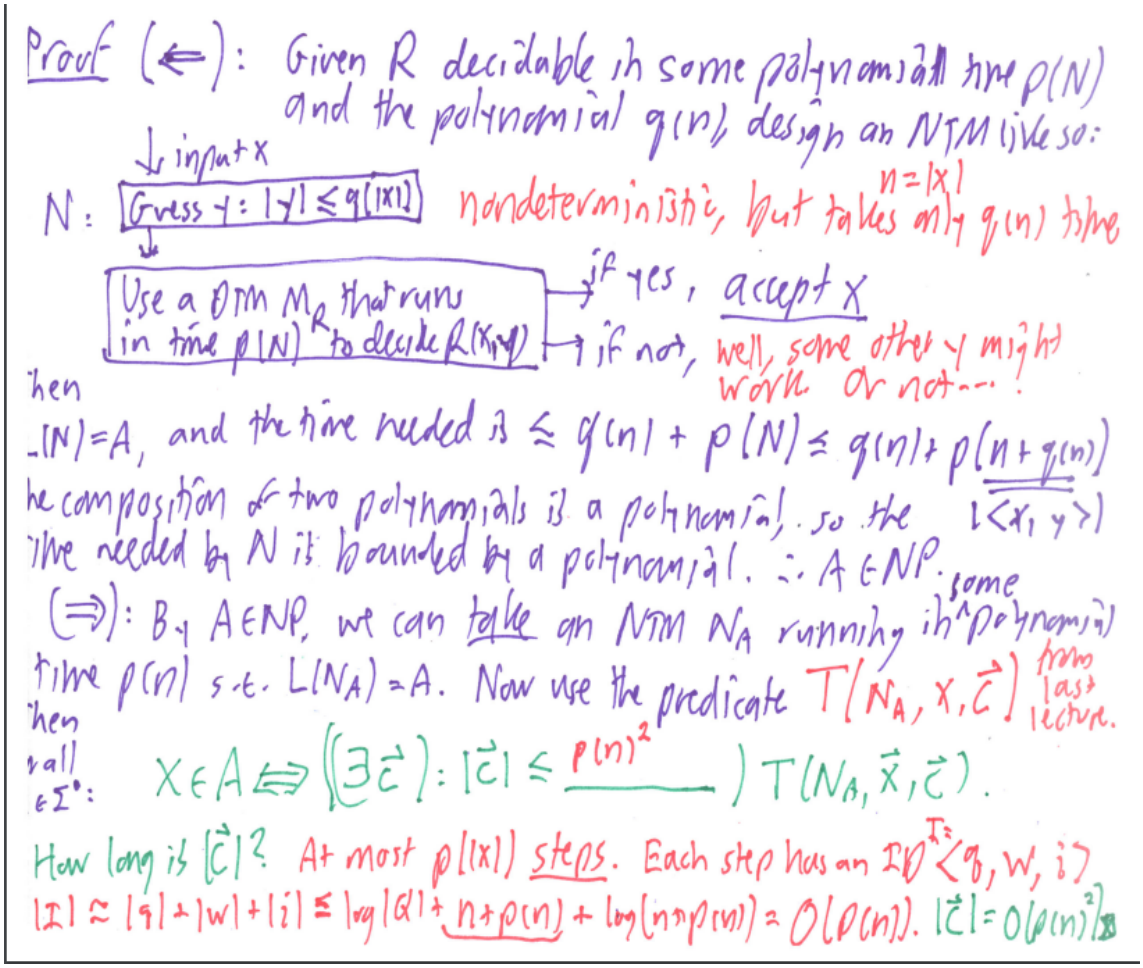
Proof. We do the $\Longleftarrow$ directions first. Given a total DTM V_R that on inputs $\langle x, y \rangle$ decides $R(x, y)$, we can build an NTM N that on input x uses nondeterminism to write down (i.e., "guess") a string y and then runs $V_R(x, y)$, accepting if and only if it accepts. Then $L(N) = L$, so L is c.e. Moreover, if V_R runs within some polynomial time bound $p(m)$, where $m = |\langle x, y \rangle| \approx |x| + |y|$, **and** if we can restrict $|y| \leq q(n)$ given x of length n, then N runs within time $p(n + q(n))$ which is also a polynomial. Thus, in the second case, L belongs to NP.

For the $\Longrightarrow$ directions, we start by taking a TM M such that $L(M) = L$ or an NTM N such that $L(N) = L$ and N runs in time bounded by some polynomial $p(n)$, respectively. The strings y we need are encodings of sequences of IDs giving valid computations $\vec{c}$ on input x by M or N, respectively. Thus we have

$$R(x, y) =_{def} T(M, x, y) \text{ or } R(x, y) =_{def} T(N, x, y), \text{ respectively.}$$

The final point is that since N also runs within space $p(n)$, the length of $\vec{c}$ for halting computations will be at most $q(n) \approx p(n)^2$. Thus in the latter case we get that for all x:

$$x \in L \iff (\exists y \in \Sigma^* : |y| \leq q(|x|)) T(N, x, y). \quad \boxtimes$$



The second part of the theorem is often used as the *definition* of NP. The polynomial-time DTM V_R is called a **verifier**, and given $x \in L$, any y such that $R(x, y)$ and $|y| \leq q(|x|)$ is called a **witness** (or **certificate**) **for** $x \in L$. 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:

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.

G3C:
Instance: An undirected graph $G = (V, E)$.

Question: Does there exist a 3-coloring of the nodes of G?

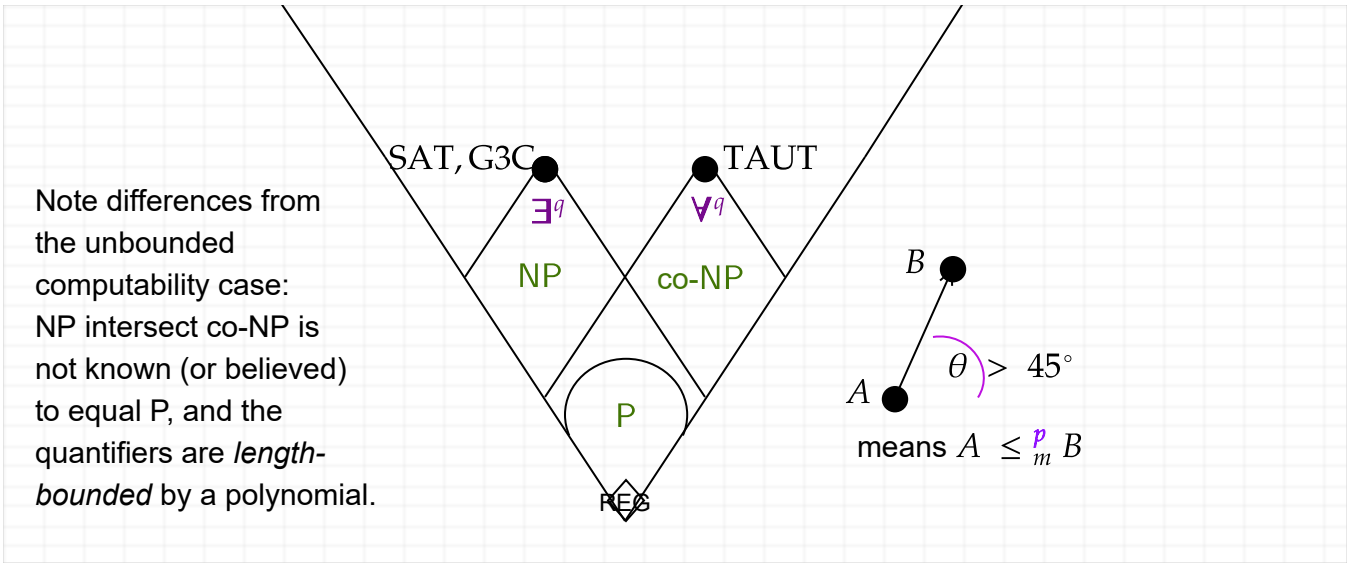
A *3-coloring* is a function $\chi : V \to \{R, G, B\}$ such that for all edges $(u, v) \in E$, $\chi(u) \neq \chi(v)$. The table for χ needs only n entries where $n = |V| \ll N = |G|$, so it has length at most linear in the encoding length N of G (often $N \approx n^2$). And it is easy to *verify* that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

Corollary: For any language L',

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

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.



[Next: more problems in these classes, polynomial-time many-one reductions, and completeness.]