Lectures and Reading. The course now turns to complexity and lectures will follow chapters 2–4 of Arora-Barak largely in sequence, with excursions into parallel coverage in the Allender-Loui-Regan notes. For Friday, please focus on the following:

- Section 2.1 of Arora-Barak, especially Theorem 2.6.
- Section 3 of ALR chapter 27 on “Circuits.”

The upshot is that if you have an NTM $N$ that runs in time $t(n) = n^{O(1)}$ then you get the following two things:

- A DTM $M$ such that for all $n$ and $x \in \Sigma^n$, $x \in L(N) \iff (\exists y \in \{0, 1\}^*)[M$ accepts $\langle x, y \rangle$ within $t(n)$ steps. Note that any parts of $y$ after the first $t(n)$ bits cannot be read in this time and would be useless, so we may suppose $|y| \leq t(|x|)$, and there is no loss of generality in making that equal, i.e., quantifying $(\exists y \in \{0, 1\}^{t(n)})$.

- A sequence of Boolean circuits $C_n$, each of size $O(t(n)^2)$, such that for all $n$ and $x \in \Sigma^n$, $x \in L(N) \iff (\exists y \in \{0, 1\}^{t(n)})[C_n(x, y) = 1]$.

The second item can be applied to any DTM $M$—multi-tape or single-tape—and carried the message that “Software Can Be Efficiently Burned Into Hardware.” By default we may suppose that $x$ too is binary, but the circuit elements still have to handle the bigger work alphabet $\Gamma$ of $M$. Still, we can re-code $\Gamma$ into $\{0, 1\}^k$ for some $k$ the way ASCII works, so all the Boolean logic can ultimately be binary, with $C_n$ consisting entirely of NAND gates.

These circuits will be instrumental next week to giving the shortcutted version of the Cook-Levin proof in ALR ch. 28, rather than Arora-Barak’s longer “traditional” Turing machine-based proof. Lurking in the background is the theorem proved at the end of Arora-Barak chapter 1 (just skim it), whose impact is to reduce the size of the circuits $C_n$ from $O(t(n)^2)$ to $O(t(n)\log t(n))$. This in turn reduces the running time of the Cook-Levin reduction function itself from quadratic (in $t(n)$ that is) to quasi-linear.

1) Suppose we have a three-string predicate $R(u, y, z)$ that is decided by a total TM $M_R$. Define the language

$$L_R = \{ u : (\forall y \in \Sigma^*)(\exists z \in \Sigma^*) R(u, y, z) \}.$$ 

For example, given a machine $M_u$, input $x$, and sequence $z$ of IDs of $M$ beginning with the starting ID $I_0(x)$, we can decide whether $z$ represents an entire accepting computation of $M_u$ on input $x$, which becomes the predicate $R(u, x, z)$. Then $L_R$ becomes $\{ u : L(M_u) = \Sigma^* \}$, i.e., the $ALL_{TM}$ language.

Show that for any decidable predicate $R$, the language $L_R$ always many-one reduces to $ALL_{TM}$. (Together with the example, this shows that $ALL_{TM}$ is complete for the class of languages that arise from decidable predicates in this “for-all there-exists” manner. 24 pts.)

2) For each of the following decision problems, say whether its language $L$ is known to belong to NP or to co-NP. In each case, say what the “witness” is, either for membership in $L$
or in \( \tilde{L} \). Give a polynomial bound on the size of the witness in terms of the input length, and briefly explain why a given witness can be checked in polynomial time. (The I/O alphabet \( \Sigma \) always includes at least two characters. In many cases, “polynomial” can be “linear.” The problems are named after films from the year 2000.)

(a) “Unbreakable”

**Instance:** An undirected graph \( G \) with an even number \( n \) of vertices, and a number \( k \) such that \( 0 < k < (n^2 - n)/2 \).

**Question:** Is it possible to break \( G \) into two disjoint pieces of \( n/2 \) nodes each by removing at most \( k \) edges?

(b) “Pay It Forward”

**Instance:** A nondeterministic Turing machine \( N \) with three-way branching at each step, an input \( x \in \Sigma^* \) to \( N \), and a number \( d > |x| \) given in unary notation as the string \( 0^d \).

**Question:** Do all possible computation paths by \( N(x) \) lead to halting and accepting within \( d \) steps?

(c) “Traffic”

**Instance:** An airport with \( k \) runways and \( n \) flights that have to land within an \( h \)-hour period. The input specifies for each flight a 30-minute time period when it can land, and which runways it can and cannot land on. No two flights may land on the same runway within fewer than 5 minutes of each other.

**Question:** Is there an air-traffic control schedule that enables all the planes to land?

(d) “Hidden Dragon”

**Instance:** A deterministic Turing machine \( M \) whose code includes a “clock” that shuts off computations on inputs of length \( n \) after \( n^2 \) steps, and a string \( y \in \Sigma^* \).

**Question:** Is there a string \( x \) of the same length as \( y \) such that \( M(x) \) outputs \( y \) and halts?

(e) “Gladiator”

**Instance:** A ’bot program \( P \) written in C that plays “Player 1” of two-player “Quake” against other ’bots that play “Player 2.” Let \( n \) be the size in bytes of the source code for \( P \).

**Question:** Does \( P \) defeat all Player 2 ’bots of the same source-code size, in “Quake” games that last at most \( n \) milliseconds and use no randomness?

You can answer the last question without knowing any details of the game “Quake” or “game-bot” programs. Depending on those details and whether ’bots can be optimized, the language of (e) may even be empty, but the point is that the above knowledge does allow you to classify it as belonging to one of NP or co-NP. The “clock” in (d) could be implemented on an independent set of tapes, after a routine that copies the input \( x \) to one of those tapes before starting “\( M \) itself.” (5 \( \times \) 6 = 30 pts., for 54 total on the set)