(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 \( \text{ALL}_{TM} \) language.

Show that for any decidable predicate \( R \), the language \( L_R \) always many-one reduces to \( \text{ALL}_{TM} \).

(Together with the example, this shows that \( \text{ALL}_{TM} \) is complete for the class of languages that arise from decidable predicates in this “for-all there-exists” manner. 24 pts.)

Answer: We need to define a computable function \( f \) such that for all \( u \), \( f(u) \) is the code of a TM \( M' \) such that \( L(M') = \Sigma^* \iff (\forall y)(\exists z) R(x, y, z) \). Taking the total TM \( M_R \) that decides \( R \), define \( M' \) as follows:

On any input \( y \):

\[
\text{for } (z = \text{emptystring}, 0, 1, 00, \ldots) \{ \\
\text{if } (M_R \text{ accepts } <u,y,z>) \{ \\
\text{accept } y; \\
\} \\
\}
\]

Building \( M' \) just involves sticking \( u \) into this finite block of code, so \( f \) is computable. For correctness, note:

\[
\begin{align*}
   u \in L_R & \implies (\forall y)(\exists z) R(x, y, z) \implies L(M') = \Sigma^* \implies f(u) \in \text{ALL}_{TM}; \\
   u \not\in L_R & \implies (\exists y)(\forall z) \neg R(x, y, z) \implies (\exists y)M'(y) \uparrow \implies L(M') \neq \Sigma^* \implies f(u) \not\in \text{ALL}_{TM}.
\end{align*}
\]

Thus \( L_R \leq_m \text{ALL}_{TM} \). Incidentally, the class of languages \( L_R \) is called \( \prod_2 \) by the analogy that the \( (\forall) \) part is some kind of infinite product, and there are two alternating quantifiers. We have shown that \( \text{ALL}_{TM} \) is complete for \( \prod_2 \). The analogy extends to give \( \prod_1 = \text{co-RE} \) and \( \prod_0 = \text{REC} \). There is a further analogy that an \( \exists \) quantifier is like an infinite OR which is like an infinite sum, so for the complementary classes we have the notation \( \Sigma_1 = \text{RE}, \Sigma_0 = \text{REC} \) since \( \text{REC} \) is closed under complements, and \( \Sigma_2 = \) the complements of languages in \( \prod_2 \). The idea extends higher to define the so-called \emph{arithmetical hierarchy}, but if we catch this at all it will be in the final weeks of class where the analogous \emph{polynomial hierarchy} will be considered.

(2) For each of the following decision problems, say whether its language \( L \) is known to belong to \( \text{NP} \) or to \( \text{co-NP} \). In each case, say what the “witness” is, either for membership in \( L \) or in \( \overline{L} \). 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?

**Answer:** “Unbreakable” is in $\text{NP}$. When the answer to an instance $(G, k)$ is “yes,” the witness can be a set of $k$ edges whose removal breaks $G$ into two equal pieces. Such a set has size at most $(n^2 - n)/2$, and the predicate “removing these $k$ edges breaks $G$ into two equal pieces” is decidable in poly-time by doing a breadth-first search from each endpoint of one of those edges, and verifying that each BFS yields $n/2$ vertices.

Logic cue: “does there exist a way to break $G$ by removing...”

(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?

**Answer:** “Pay it Forward” is in $\text{co-NP}$. When the answer is no, the witness can be a rejecting computation path. The string $y$ for such a path can be a ternary string of length at most $d$, and since “$0^d$” is part of the input, the length of $y$ is linear in the input length. To verify that $y$ represents a rejecting path, we need to simulate $N$ for $d$ steps using $y$ to resolve nondeterminism. With $N$ given as a list of tuples (or as a .tmt file in the Turing Kit), and with the input $x$ to $N$ being given as part of the instance, we need only appeal to the ability to simulate each move of $N$ by taking one left-to-right “sweep” thru the code of $N$. This takes $O(|N| \times d)$ steps to do $d$ steps of $N$, but since the input length $n \geq |N| + d$, that’s polynomial in $n$.

(If $N$ has an arbitrarily large number $k$ of tapes, then we can still simulate $N$ through the $k$-tapes-to-one simulation in class. Then we might need $2k$-many left-to-right sweeps thru the code of $N$ to update each head, plus the left-to-right-to-left sweep thru the current contents of the one “real” tape we use to simulate the $k$ “virtual tapes” of $N$. However, over $d$ steps, the contents have size at most $(|x| + d)$ multiplied by the number of bits required to represent a character in $N$’s work alphabet; since this number is certainly less than $|N|$, the contents have size at most $|N|(|x| + d)$, and we get the $d$ steps of $N$ done in time at most $d(|N|(|x| + d) + 2|N|)$, which is polynomial in $|N| + |x| + d$. It was OK not to address this technical point—one can almost always assume that a TM given as input has 2 worktapes with alphabet \{0, 1, B\}.)

Logic cue: ”...all possible computation paths...”

(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?

**Answer:** “Traffic” is in $\text{NP}$: Given $k, n, h$, and most importantly, the table of allowable landing times that is part of the instance, the witness is a workable schedule. The schedule has length roughly linear in the number of flights, hence linear in the size of the input table. Verifying that the schedule works likewise takes time roughly proportional to the number of flights, and is an easy matter of checking that all times stated by the schedule are within the stated interval for each flight, and at least 5 min. apart on each individual runway.

Logic cue: “...does there exist a schedule...”
(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?

Answer: “Hidden Dragon” is in NP: When the answer to an instance $(M, y)$ is “yes,” the witness is the string $x$ of length $|y|$ such that $M(x) = y$. The length of $x$ is (sub-)linear in $|(M, y)|$, and verification takes at most $|x|^2 = |y|^2$ steps of $M$ via the clock. (The extra parenthetical technical remarks in (b) can apply here as well, to multiply things by one or two more factors of $|M|$ and/or $|y|$, but that’s still bounded by a polynomial in $|M| + |y|$, and both here and in (d), the Hennie-Stearns Theorem (Theorem 5.6/Corollary 5.5 on p94) knocks down these overhead factors to $O(\log |M| + \log |y|)$.)

Logic cue: “...does there exist an $x$...”

(e) “Gladiator”

Instance: A bot program $P$ written in C that plays “Player 1” of a two-player game of Quake III Arena 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?

Answer: “Gladiator” as stated is in co-NP: The witness for a no answer is a gamebot program $Q$ taking Player-2 that beats $P$. The length of $Q$ is stated to be the same as that of the input $P$, hence “linear” in $n = |P|$. Verification can consist of a single run of $P$ and $Q$ on a real computer; since this takes $n$ milliseconds and a TM suffers at most a quadratic slowdown when simulating a RAM program representing the compiled code, the TM-defined runtime to verify that $Q$ beats $P$ is polynomial in $n$.

Logic cue: “...does $P$ beat all Player-2 ’bots...”