Reminders: The Final Exam is on Tuesday, May 10, in the “middle time” 11:45am–2:45pm in the lecture room, Hochstetter 114. The rules are closed book, open notes, no electronics. Ideally notes should be in one binder or folder, to minimize the space you take next to your neighbors. There is no need for a calculator. If you bring a backpack, we will ask you to stow it on the side of the room.

There is a Review Session Monday 2–4pm in Alumni 97. I will hold office hours after it until 6pm.

(1a) Prove that the following decision problem is undecidable:

**INSTANCE:** A Turing machine $M$ with input alphabet $\Sigma = \{0, 1\}$.
**QUESTION:** Does $M$ accept any string of the form $0^n1^n$ where $n \geq 1$?

First state what the language $L$ of this problem is using ordinary set-builder notation like in the text and lectures. Then prove it is undecidable by reduction from $A_{TM}$ (or from $K$ if you prefer). Finally answer: is $L$ recognizable (synonymously: recursively enumerable, computably enumerable, r.e., c.e.)? (3+18+6 = 27 pts.)

(1b) Prove that the language $\{\langle M \rangle : M$ is a DTM and $L(M) = L(M)^R\}$ is undecidable. Note that the only difference from the decidable problem (1) of PS9 is that $M$ is a deterministic TM not a DFA, but it’s a big difference. **Condition:** Your proof must already follow from the reduction construction in problem (1a). So this is worth just 6 pts. for the tweaked analysis.

**Answer:** $L = \{\langle M \rangle : L(M) \cap \{0^n1^n : n \geq 1\} \neq \emptyset\}$. It was also OK to resort to prose in the body of the set after ‘:’ by saying, “$M$ accepts some string of the form $0^n1^n$ where $n \geq 1$.”

To show $A_{TM} \leq_m L$, define a reduction function $f$ to map any instance $\langle M, x \rangle$ of the former to a machine $M'$ coded as follows: “On input $w$, first simulate $M(x)$. If and when that computation accepts, then test whether $w$ has the form $w = 0^n1^n$ for some $n \geq 1$. If so, accept $w$; else, reject $w$.”

Then:

\[
\begin{align*}
\langle M, x \rangle \in A_{TM} & \implies L(M') = \{0^n1^n : n \geq 1\} \implies f(\langle M, x \rangle) = \langle M' \rangle \in L \\
\langle M, x \rangle \notin A_{TM} & \implies L(M') = \emptyset \implies f(\langle M, x \rangle) = \langle M' \rangle \notin L
\end{align*}
\]

Since $A_{TM}$ is undecidable, $L$ is undecidable—and also not co-c.e. Is $L$ c.e.? The answer is yes. Among several ways to see this, the crispest is to note that the same $M'$ belongs to $NE_{TM}$ if and only if the code of $M$ is in $L$. Hence $f$ also reduces $L$ to the $NE_{TM}$ language covered in lecture which is c.e. By the lemma that $B \in \text{RE} \wedge A \leq_m B \implies A \in \text{RE}$, we have $L \in \text{RE}$.

(b) And $0^R = \emptyset$ is reversible but $\{0^n1^n : n \geq 1\}$ is not, so $f$ also reduces $A_{TM}$ to the complement of $L_b\{\langle M \rangle : M$ is a DTM and $L(M) = L(M)^R\}$. Since the class $\text{REC}$ is closed under complements, $L_b$ too cannot be decidable.

Footnotes: If you were to interpret “any” as “all” then the language would be $L'' = \{\langle M \rangle : \{0^n1^n : n \geq 1\} \subseteq L(M)\}$ instead. However, the above $f$ reduces $A_{TM}$ to $L''$ with exactly the same analysis—no change to anything. (It’s just like we specialized the “all-or-nothing switch” to $\{0^n1^n : n \geq 1\}$.) However, $L''$ is not c.e. The “Delay Switch” technique works fine to get a reduction from $D_{TM}$ to $L''$: Make the function $g$ map an instance $\langle M \rangle$ to a TM $M''$ that behaves as follows: “On any input $w$, take $t = |w|$ and run $M$ on its own code for up to $n$ steps. If $M$ did not accept its own code within
that time, then test whether \( w \) has the form \( w = 0^n1^n \) for some \( n \geq 1 \), and accept \( w \) iff the answer is yes. But if \( M \) did accept \( \langle M \rangle \) within the \( t \) steps, then reject \( w \) out-of-hand. Then the analysis is:

\[
\langle M \rangle \in D_{TM} \quad \Rightarrow \quad L(M') = \{0^n1^n : n \geq 1\} \quad \Rightarrow \quad g(\langle M \rangle) = \langle M'' \rangle \in L''
\]

\[
\langle M \rangle \notin D_{TM} \quad \Rightarrow \quad L(M') \text{ is finite} \quad \Rightarrow \quad \{0^n1^n : n \geq 1\} \notin L(M') \quad \Rightarrow \quad \langle M' \rangle \notin L''.
\]

Thus \( A_{TM} \leq_m L'' \land D_{TM} \leq_m L'' \), which is a foolproof recipe for showing that \( L'' \) is neither c.e. not co-c.e. (It is in fact mapping-equivalent (\( \equiv_m \)) to ALL_{TM} but trying for that isn’t foolproof in other cases.) Finally it is worth noting that these reductions also suffice to show that REGULAR_{TM} is undecidable, although the proof varies the order of the tests from the text’s proof of this.

(2) Consider the text’s algorithm for deciding the \( A_{NFA} \) problem in Theorem 4.2. As a language, \( A_{NFA} = \{ \langle N, w \rangle : N \text{ is an NFA and } w \in L(N) \} \). Let \( n = |w| \) and let \( m \) be the number of states in \( N \), so that the overall length of the input \( x = \langle N, w \rangle \) is \( \text{roughly} \ (\text{order-of}) \ m + n \). Now the text’s algorithm can take time order-of \( n2^m \) in worst case because the NFA \( N \) might blow up to a DFA of about \( 2^n \) states. Work out a different algorithm that decides the problem in time roughly \( O(\text{mmn}) \) or at worst \( O(m^2n) \). Here “roughly” means that you may ignore \( O(\log m) \) factors—that is, treat each state of \( N \) as a unit although it really has a binary-number label of length \( O(\log m) \). Since this is quadratic or at worst cubic in \( m + n \), what you have done is classify the \( A_{NFA} \) problem into deterministic polynomial time. (Think of how you would actually simulate all the possible states as \( w \) is read bit-by-bit. 12 pts.)

Answer: Given the encoding of the NFA \( N = (Q, \Sigma, \delta, s, F) \) and its input \( x \in \Sigma^* \), taking \( n = |x| \) and \( m = |Q| \), our algorithm \( A \) avoids the NFA-to-DFA blowup by simulating \( N \)'s possibilities on \( x \) directly. Our \( A \) maintains the sets \( S_i \) of states \( N \) could be in upon processing the \( i \)-th bit of \( x \). Initially \( S_0 \) equals the set \( E(s) \) containing \( s \) plus all states reachable by \( \epsilon \)-arcs (if \( N \) has any) starting from \( s \). This is in fact the start state of the equivalent DFA, but we’re not building all up-to-\( 2^m \) states of that DFA, just the up-to-\( n \) other states we may encounter while processing \( x \). Note that this process is breadth-first search just like in the algorithm for \( E_{DFA} \) but traversing only \( \epsilon \)-arcs, so it runs in polynomial time \( O(m^2) \) time at worst. Now suppose we have the right set \( S_{i-1} \) before reading the char \( c = x_i \). We loop through the up-to-\( m \) states \( q \in S_{i-1} \) and collect all \( r \) such that \( (q, c, r) \in \delta \) into a set \( S' \). This needs at worst \( m \) passes through the code of \( N \) for each \( q \), so adds another \( O(m^2) \) time at worst. Finally we use \( \epsilon \)-closure again to get \( S_i = E(S') \) taking a third dose of \( O(m^2) \) time. This adds up to \( O(m^2) \) time for each \( x_i \), which makes \( O(nm^2) \) time overall.

Footnotes If one makes the reasonable assumption that every state in \( N \) has at most 2 instructions and connects to at most 2 other states by any chain of \( \epsilon \)-arcs, then one can get \( \tilde{O}(n) \) time even on a two-tape Turing machine. This involves coding \( N \) so that states are in sorted order with all instructions for the same state together and likewise sorting the sets \( S_i \) obtained at each stage. You may have seen a similar sorting trick used to tell whether all items in a list or vector are unique in CSE250 or CSE331. But this extra efficiency doesn’t matter for showing the problem is in \( P \)—even \( O(nm^4) \) time is “polynomial enough.” The attitude for a long time has been: “just get me some polynomial so I know the problem scales—then we can work on optimizing it.” However, optimum efficiency does matter insofar as this is how UNIX and languages like Perl and Ruby actually match lines \( x \) of text to regular expressions \( r \) that you code: first \( r \) (or \( \Sigma^* r \Sigma^* \) if the match is anywhere on the line) is converted to an equivalent NFA \( N \) by pretty much the construction in the text (with shortcuts noted in lecture), then \( N \) is executed on \( x \) as above.

(3) Now consider the following problem which is not quite the same as what the text’s general notation scheme would call “ALL_{NFA}”:

**Instance:** An NFA \( N = (Q, \Sigma, \delta, s, F) \) and a number \( n \), where \( 1 \leq n \leq |Q| \).

**Question:** Is \( \{0,1\}^n \subseteq L(N) \)?
This is just asking whether $N$ accepts all binary strings of the given length $n$, not whether it accepts all strings as with \textit{"ALL$\overline{NFA}$"}.

(a) State the language $L$ of this problem and also state its complement $L'$—ignoring inputs that are not valid codes $\langle N, n \rangle$. (6 pts.)

(b) Show that $L'$ belongs to NP. That is, use your answer to problem (2) to give a polynomial-time verifier for the cases where $N$ \textit{fails} to accept some string of length $n$. (6 pts.)

(c) Show that $L$ polynomial-time many-one reduces to \textit{ALL$\overline{NFA}$}. Given $N$ and $n$, you need to create an NFA $N'$ that also accepts all strings of length strictly less than $n$ and all strings of length strictly more. Sketch abstractly how to build $N'$ to accomplish the reduction. (9 pts., for 21 on the problem and 66 on the set; the language $L'$ is in fact NP-complete.)

Answer: $L = \{\langle N, n \rangle : N \text{ is an NFA and } \{0, 1\}^n \subseteq L(N)\}$. The essential complement is $L' = \{\langle N, n \rangle : N \text{ is an NFA and } \{0, 1\}^n \not\subseteq L(N)\}$. It is not the literal complement because of cases where $N$ is not an NFA plus strings not in the range of $\langle \cdot, \cdot \rangle$ at all. It is, however, the complement of $L$ within the Type \textit{“An NFA and a natural number”} and this is good enough.

(b) Whenever $\langle N, n \rangle \in L'$, i.e., whenever $\{0, 1\}^n \not\subseteq L(N)$, it means that there exists a string $x \in \{0, 1\}^n$ that $N$ does not accept. An NTM $U$ needs only $n$ steps to guess such an $x$ and then can verify $\langle N, x \rangle \notin A_{\overline{NFA}}$ in polynomial time using your (deterministic!) algorithm in problem (2). This embodies the definition of $L'$ belonging to NP.

(c) To create $N'$, we use our old $\epsilon$-fork trick at a new start state $s'$. One of the $\epsilon$-arcs goes to the start state $s$ of the NFA $N$ given in the argument to the reduction function $f$ we’re coding. The other goes to the first of $n + 2$ states $q_0, \ldots, q_{n-1}, q_n, q_{n+1}$ that each have arcs on both 0 and 1 to the next state in the series. The first $n$ of these states are accepting, which makes $N'$ able to accept all strings of lengths 0 to $n-1$. The next one is \textit{not} accepting, which keeps $N'$ from accepting any additional strings of length $n$. The last one is not only accepting but also a \textit{“nirvana state”} that loops back to itself, which accepts all strings of length $n + 1$ and higher. Thus $L(N') = \Sigma^* \iff \{0, 1\}^n \subseteq L(N)$. The code for $N'$ requires only $O(n)$ additional states and arcs with easy wiring on top of those already given in $N$, so $f(\langle N, n \rangle) = \langle N' \rangle$ is computable in polynomial time—indeed, linear time even without ignoring log($n$) factors. So $L \subseteq_m \text{ALL$\overline{NFA}$}$.  

Footnote: We can in fact show 3SAT $\leq_m L'$ by the following construction. Given a CNF formula $\phi$ with clauses $C_1 \land \cdots \land C_m$, make an NFA $N$ with not just 2 but $m$ $\epsilon$-arcs at its start state. Each goes to a series of $n + 1$ states for the corresponding clause $C_j$ that is like the one in part (c) but different. The difference is that if $x_i$ occurs in $C_j$, then we take away the arc $(q_{i-1}, 1, q_i)$, while for any negated literal $\overline{x}_k$, we take away $(q_{k-1}, 0, q_k)$ instead. Thus a binary string $w = a_1, \ldots, a_n$ can make it to the nirvana state $q_n$ at the end of the series if and only if it \textit{violates} all the literals in the clause $C_j$, meaning that the corresponding truth assignment \textit{fails} to satisfy the clause. The upshot is that there is a string $w \in \{0, 1\}^n$ that $N$ fails to accept iff there is such a $w$ that “fails to violate” any of the clauses—which is another way of saying that $w$ \textit{satisfies} all the clauses, which means $\phi \in 3SAT$. The arcs of $N$ can be built straightforwardly in $O(nm)$ time ingoring log factors, so the function $f(\phi) = N$ is a polynomial-time reduction from 3SAT to $L'$, thus $L'$ is NP-complete. Together with (c) this also makes \textit{ALL$\overline{NFA}$} hard for \textit{co-NP} under polynomial-time mapping reductions. Which makes a nice counterpart to the fact that \textit{ALL$\overline{CFG}$} is \textit{undecidable}—although \textit{ALL$\overline{NFA}$} is decidable there is no \textit{“perfect TA”} who can grade every NFA you $N$ you submit on homework in polynomial time—unless you yourself supply a \textit{proof} that $L(N) =$ the problem’s target language $E$. Proofs are checkable in polynomial time, and this finally says why it is important to have them for both grammars and (N)FAs.