(1) For any language \( L \), define \( \text{Halves}(L) \) to be the language of prefixes \( w \) of strings \( x \in L \) s.t. \( w \) includes at least the first half of \( x \). (In symbols, this gives \( x \in L, w \subseteq x, \) and \( |w| \geq |x|/2 \).) Also define \( \text{Prefs}(L) \) to be the set of all prefixes of strings in \( L \), i.e. without the \(|w| \geq |x|/2 \) restriction.

(a) Prove that if \( L \) belongs to \( \mathbb{P} \), then \( \text{Halves}(L) \) belongs to \( \mathbb{NP} \).

(b) Prove however that when \( L \) is a suitable encoding of the valid-computation checking predicate (the “Kleene \( T \) predicate”) then the language \( \text{Prefs}(L) \) is undecidable, even though \( L \) belongs not only to \( \mathbb{P} \) but even to deterministic linear time (on a 2HDFA in fact).

(c) Also answer: Is \( \text{Prefs}(L) \) always c.e. when \( L \) is c.e.? (6 + 12 + 6 = 24 pts.)

\[ \text{Answer:} \]

(a) For all \( w, w \in \text{Halves}(L) \iff (\exists x : |x| \leq |w|)[wx \in L] \). The bound \( |x| \leq |w| \) must hold because taking \( z = wx \) we have \(|w| \geq |z|/2 \), so \(|x| \leq |z|/2 \) since \( x \) is the rest of \( z \), so \(|x| \leq |w| \). The part \([wx \in L]\) is a polynomial-time decidable predicate. Thus \( \text{Halves}(L) \) is defined by a polynomial (indeed linear) length-bounded \( \exists \) quantifier on a polynomial-time verifier, so it belongs to \( \mathbb{NP} \).

(b) With \( \text{Prefs}(L) \) we don’t have the length bound on \( x \) anymore. However, not only does \( \text{Prefs}(L) \) no longer belong to \( \mathbb{NP} \) it can be undecidable. Consider \( L = \{ \langle M, w, \vec{c} \rangle : \vec{c} \) is a valid accepting computation of \( M \) on input \( w \} \), where the angle brackets and commas are literal characters (that won’t occur in the \( M, w, \) or \( \vec{c} \) parts). Then \( L \) is decidable—it’s just an encoding of the \( T \)-predicate language as mentioned in the problem. But now consider strings of the form \( \langle M, w, \rangle \), where again the commas are literal. They belong to \( \text{Prefs}(L) \) if and only if \( M \) accepts \( w \). Thus the ability to decide \( \text{Prefs}(L) \) entails being able to decide \( A_{TM} \). Put another way, \( A_{TM} \leq_m \text{Prefs}(L) \) via the function that simply changes the closing angle bracket in \( \langle M, w \rangle \) into a comma. So even though \( L \) itself is decidable in linear time, \( \text{Prefs}(L) \) is undecidable.

(c) We still get that \( \text{Prefs}(L) \) is c.e. when \( L \) is c.e.: if we have a decidable predicate \( R(x, y) \) such that \( x \in L \iff (\exists y) R(x, y) \), then we get \( w \in \text{Prefs}(L) \iff (\exists x, y) R(wx, y) \), which still has the required \( \exists \cdot \text{REC} \) form.

(2) A (ternary) mask is a string whose entries are either 0, 1, or \(* \) for “don’t care.” A string \( x \in \{0, 1\}^* \) matches a mask \( w \) if for all bit places \( i, 1 \leq i \leq n \), either \( x_i = w_i \) or \( w_i = * \). If \( x \) does not match a mask \( w \), let us say that \( x \) “violates” \( w \). For example, if \( w = *01*1* \), then 001110 matches \( w \) but 001001 violates \( w \) in place \( i = 5 \). Show that the following decision problem is \( \mathbb{NP} \)-complete:

\[ \text{MASKS} \]

\[ \text{INSTANCE:} \] A finite set \( w_1, \ldots, w_m \) of masks, each of the same length \( n \geq 1 \).

\[ \text{QUESTION:} \] Does there exist a string \( x \in \{0, 1\}^n \) that violates all the masks?

Now go on to replace the “don’t care” star by the regular expression \((0 + 1)\). Then regard your sets \( w_1, \ldots, w_m \) of masks as a single regular expression \( r \), one that uses \( + \) and \( \cdot \) but not Kleene star, so that \( L(r) \) is finite. Conclude that the following problem is also \( \mathbb{NP} \)-complete:

\[ \text{INSTANCE:} \] A regular expression \( r \) and the number \( n \) (in the form \( 0^n \)).

\[ \text{QUESTION:} \] Does there exist a string \( x \in \{0, 1\}^n \) that is not in \( L(r) \)?
Finally conclude that the problem “\textsc{AllShort}_{NFA}” in the Wed. 10/24 lecture is co-NP-complete—namely, that the complement of its language is NP-complete. (18 + 9 + 9 = 36. pts.)

\textbf{Answer:} (a) For the “in NP” part, whenever the answer to a given finite set \( S \) of masks is “yes,” we can \textit{guess} \( x \) and \textit{verify} that each \( w \in S \) has a bit-conflict with \( x \), in time linear (hence “polynomial”) in the size of \( S \).

To show it is complete, we show \( 3\text{SAT} \leq_{p}^{n} \text{MASK} \). Let any 3CNF formula \( \phi \) in \( n \) variables with \( m \) clauses be given. For each clause \( C \) build the string \( w \) entries \( w_i \) defined by:

\[
\text{if } x_i \in C \text{ then } w_i = 0 \text{ else if } \bar{x}_i \in C \text{ then } w_i = 1 \text{ else } w_i = *
\]

for \( i = 1, \ldots, n \). (If both \( x_i \) and \( \bar{x}_i \) occur in \( C \) then skip clause \( C \) since it’s always satisfied. Even better, you can ignore this possibility because lectures showed \( 3\text{SAT} \) is NP-complete via formulas whose clauses contain no redundant literals.) Then \( C \) is satisfied by exactly those assignments \( x \in \{0,1\}^n \) that mismatch \( w \) in some place \( i \) where \( w_i \neq * \). Hence \( \phi \in 3\text{SAT} \iff \text{there is an } x \text{ that satisfies all clauses in } \phi \iff \text{there is a } x \text{ that violates all } w \in S \iff S \in \text{MASK}, \text{ where } S \text{ is the set of strings } w \text{ thus obtained. Thus the function } f(\phi) = S \text{ reduces } 3\text{SAT} \text{ to } \text{MASK}, \text{ and it is computable in time } O(nm), \text{ polynomial in } |\phi| \).

(b) Upon replacing every * in our masks \( w_j \) by \((0+1)\) to get a regular expression \( r_j \), and unioning together all the \( r_j \) to get \( r \), we need to flip around the interpretation again: A string \( x \) that matches \( r_j \) does \textit{not} violate the mask. Since violating strings satisfied the clause \( C_j \), this means \( x \) does \textit{not} satisfy \( C_j \). Hence \( L(r_j) \) is the set of strings \( x \in \{0,1\}^n \) that do not satisfy \( C_j \). The union of those sets, which is \( L(r) \), thus equals the set of strings that fail to satisfy at least one clause. Thus \( L(r) = \{0,1\}^n \) if and only if every string fails to satisfy at least one clause—that is, iff \( \phi \) is unsatisfiable. So there exists a string \( x \in \{0,1\}^n \) that is not in \( L(r) \) if and only if \( \phi \) is satisfiable. Thus we have reduced \( 3\text{SAT} \) to problem (b).

To show that problem (b) belongs to NP in general—not just that the cases in the reduction can be easily verified—we need to argue that non-membership of \( x \) in \( L(r) \) for a general regular expression can be decided in polynomial time. This is doable because the conversion from regular expressions \( r \) to NFAs \( N_r \) shown in class is an efficiently computable simple recursion, and then we can apply the polynomial-time procedure for the \( A_{NFA} \) problem shown in class.

(c) Finally, to reduce problem (b) to the complement of the “\textsc{AllShort}_{NFA}” language (which belongs to NP for the same reason), we apply that very same conversion. Here we can use the particularly simple form of regular expressions \( r = r_1 + \cdots + r_j + \cdots + r_m \) which we obtained in (b). The NFA \( N_r \) has a giant branch of \( \epsilon \)-arcs from its start state to starts \( s_j \) for components \( M_j \) for each \( r_j \). The component \( M_j \) is actually deterministic: it has a simple progression of states \( s_j = q_0,j, q_1,j, q_2,j, \ldots, q_{n-1},j, q_n,j \) with \( q_n,j \) accepting. For each \( i \), there are two arcs from \( q_{i-1},j \) on \( 0 \) and 1 into \( q_i,j \) if the \( i \)-th part of \( r_j \) is the “don’t care” \((0+1)\), else it has just a single arc for the bit that is there. Thus \( N_r \) doesn’t even have any cycles, and yet it is co-NP-complete to tell whether \( L(N_r) = \{0,1\}^n \).

(3) Write a formal mathematical definition of the language of the following decision problem, using notation such as \( G = (V, E) \) and edges as \( (u, v) \) where \( u, v \in V \). Use the definition to show that the language of the following decision problem belongs to NP (6. pts.). Then show that the problem is NP-complete \textit{by reduction from} \( 3\text{SAT} \). (30 pts., making 36 on the problem and 96 on the set)

\textbf{Edge Cover}

\textbf{Instance:} An undirected graph \( G \), an integer \( k \geq 1 \).

\textbf{Question:} Does there exist a set \( H \) of at most \( k \) edges such that every other edge in \( G \)
touches an edge in \( H \)? (Two edges “touch” if they share a vertex.)
Answer: In full formality, where $G = (V, E)$ is understood, we have:

$$L_{EC} = \{ G\#k : (\exists H \subseteq E)[|H| \leq k \land (\forall (u, v) \in E)(\exists (t, w) \in H)[\{u, v\} \cap \{t, w\} \neq \emptyset]] \}.$$ 

Anything close to this could be full credit. Now only the initial $(\exists H \subseteq E)$ is a “heavy” quantifier over potentially exponentially many subsets—the other two are over the polynomially-many edges in the graph, so they can be ignored. Hence this is an $\text{NP}$-definition of $L_{EC}$.

For hardness under poly-time many-reductions, we reduce 3SAT to $L_{EC}$. Given any $3\text{CNF}$ formula $\phi = C_1 \land \cdots \land C_m$ with $n$ variables, we build the graph $f(\phi) = G_{\phi}$ as follows: We allocate the usual “rung” vertices $x_i, \overline{x}_i$ and “clause triangles” as before, but we also allocate some extra vertices. Between every pair $x_i, \overline{x}_i$ we insert a node $u_i$ to make the edges become $(x_i, u_i)$ and $(u_i, \overline{x}_i)$.

To “make assurance double sure” as Macbeth says in Shakespeare’s Scottish play, we also allocate nodes $v_i$ connected only to $u_i$. This makes totally clear that to cover the edge $(u_i, v_i)$, one will need to choose one of the two edges $(x_i, u_i)$ or $(u_i, \overline{x}_i)$, since there is no way to cover it from outside.

Thus each rung looks like a ‘T’ with four nodes, where the leg of the T is the extra edge we added. Without loss of generality there is never any reason to choose the leg, since each of the two top edges covers at least as many other edges. Hence we can assert that any minimum edge cover must choose one of the two top edges in any T. The choices of these edges are thus in 1-to-1 correspondence with truth assignments.

To cover the clause gadgets we need at least one edge per triangle. Hence $k := n + m$ is the absolute minimum possible size for any edge cover. The “crossing edges” go from any $x_j$ in $C_j$ to $x_i$ in the rung, and from any $\overline{x}_i$ in $C_j$ to $\overline{x}_i$ in the rung. This completes the construction, and its time complexity is clearly polynomial in $n$ and $m$.

For correctness, we claim the target $k$ is attainable iff $\phi$ is satisfiable. First, if $a$ is a satisfying assignment to $\phi$, we can determine the corresponding minimum-size edge cover $H_a$ by first including the rung edges corresponding to the assignment. Since $a$ satisfies $\phi$, ever clause has at least one of its crossing edges already covered. Hence choosing the edge in the triangle that is opposite that crossing edge covers the other two crossing edges, as well as the other two edges within the triangle. Hence our $H_a$ of size $n + m$ covers all edges.

Going the other way, suppose we have an edge cover $H$ of size $m + n$. Now a fine point is that it is possible to cover a clause triangle without choosing one of its internal edges, by choosing two crossing edges into it instead. However, doing so “wastes” an edge, and makes it impossible to complete an edge cover of the minimum possible size $m + n$ when you consider the rungs and the other clauses. Indeed, just choosing one crossing edge wastes it. Hence $H$ must consist solely of $m$ edges within the $m$ clause triangles (one in each) and $n$ in the rungs. The only way a single triangle edge in a clause gadget can work is if the crossing edge from the opposite vertex of the triangle is covered by the rung it goes to. This means that the corresponding clause is satisfied by the truth assignment from the rungs. Since this happens for each clause, $\phi$ must be satisfied by the truth assignment made in the rungs. Hence the reduction is correct.