(1) Prove by reduction from $A_{TM}$ (or $K_{TM}$) that the following decision problem is undecidable:

**IF-ELSE**

**INSTANCE:** A program $P$ in Java (or some other high-level programming language) and a particular if-then-else statement $S$ in $P$.

**QUESTION:** Does there exist an input $x$ such that when $P(x)$ is run, there is a point in the computation where the “else” branch is taken?

*Important note:* it is assumed that the condition between if and then is decidable—i.e., that the if-else statement itself always exits, as required of a flowchart diamond.

**Also answer:** is the language of the problem c.e.? Justify briefly. (12+6 = 18 pts.)

**Answer:** There are several answers ranging from the “waiting for Godot” idea through the “all or nothing switch” idea, with the last one more like the “delay switch” idea. All of them define a reduction function $f(M, w) = (P, S)$ where $P$ is a program and $S$ is a designated if-(then-)else statement in $P$. The first one makes the if-else statement trivial but not in a way that violates its having a decidable test (per the added note).

Call the language of this problem $IE$. To put the second part of the question, $IE$ is c.e. because we can design a Java interpreter $J$ that given $P$ and $S$ runs a governing unbounded loop over $t = 1, 2, 3, 4, \ldots$. In the body of the loop it tries $P(x)$ for all $x \leq t$ (or: for all $x$ of length up to $t$) for up to $t$ steps. If and when the interpreter ever sees the else of $S$ branch being taken, it accepts $⟨P, S⟩$. Thus $L(J) = IE$, so the language IE is c.e. Alternatively, we can show that $IE$ is c.e. by breaking down the logic of its definition:

$$⟨P, S⟩ \in IE \equiv (∃x)(∃t)[P(x) \text{ executes the else-branch within } t\text{steps}].$$

The predicate in $[\ldots]$ is decidable, so the language is defined by a purely existential quantification over a decidable predicate. That puts the language into RE.

For the four reduction answers (that is, the four I had in mind while writing the problem), we label $P$ as $P_1, P_2, P_3, P_4$ for the four respective answers. In the first possible answer, $P_1$ works as follows on any input $x$:

1. Simulate $M(w)$ open-endedly, so that control moves on only if $M$ accepts $w$ (you can convert rejection into never-halting if you wish).

2. $S$: if (false) goto 1; else accept $x$;

Then $⟨M, w⟩ \in A_{TM} \implies$ for all $x$, $P_1(x)$ reaches the if-else statement $S$ and takes the “else” branch. Thus there exists such an $x$, so $f(M, w) = (P_1, S) ∈ IE$.

Whereas, $⟨M, w⟩ \notin A_{TM} \implies$ for all $x$, $P(x)$ never even reaches the if-else statement $S$. So there certainly does not exist an $x$ such that it takes the “else” branch of $S$, so $(P_1, S) \notin IE$.

The second answer doesn’t need you to do anything special about the case where $M(w)$ halts and rejects—it just gets taken case of naturally. The program $P_2$ is:
1. Simulate $M(w)$ open-endedly, so that control moves on only if $M(w) \downarrow$.

2. $S$: if $(M$ rejected $w)$ reject $x$; else accept $x$;

Then $\langle M, w \rangle \in A_{TM} \implies$ for all $x$, $P_2(x)$ reaches the if-else statement $S$ and takes the “else” branch. Thus there exists such an $x$, so $f(M, w) = (P_2, S) \in IE$.

Whereas, $\langle M, w \rangle \notin A_{TM} \implies$ for all $x$, $P_2(x)$ either never reaches the if-else statement $S$, or it reaches and takes the “then” branch. Either way, $(P_2, S) \notin IE$.

The other two answers break the simulation of $M(w)$ into individual steps, though the first one still does not use the input $x$ to limit it and still ignores $x$, so it has more the character of the “all-or-nothing switch.” The program $P_3$ is:

1. Do one (more) step of the computation $M(w)$ (unless $M$ has not already halted, in which case, fall through).
2. $S$: if $(M$ did not accept $w$ in that step) goto 1; else accept $x$;

This $P_3$ has the property that the if-else statement is always reached at least once—the question is how it branches. Now the analysis is:

$\langle M, w \rangle \in A_{TM} \implies$ for all $x$, $P_3(x)$ eventually stops taking the “then” branch and takes the “else” branch, which is when it accepts $x$. So there certainly exists such an $x$, so $(P_3, S) \in IE$. Whereas:

$\langle M, w \rangle \notin A_{TM} \implies$ for all $x$, $P_3(x)$ always takes the “then” branch. Thus there does not exist an $x$ that makes $P_3(x)$ take the “else” branch, so $(P_3, S) \notin IE$.

In all of these cases it is clear that once we drop $M$ and $w$ into designated places in the pseudocode we get a definite program $P$ and statement $S$, so that mapping $f(M, w) = (P, S)$ is (easily) computable. The fourth answer $P_4$ uses its own input $x$ to “meter” the simulation of $M(w)$ as in the “delay switch” idea:

1. On input $x$, set $n = |x|$.
2. Do up to $n$ steps of the computation $M(w)$.
3. $S$: if $(M$ did not accept $w$ in that time) reject $x$; else accept $x$;

This $P_4$ has the property that it always flows straight-through and hits the if-else statement once and only once. Thus the branch it takes is a one-shot deal. The analysis is still open-ended because there are infinitely many possible $x$-es, and this is finally where “there exists an $x$” has separate significance from “for all $x$.” The analysis is:

$\langle M, w \rangle \in A_{TM} \implies M$ accepts $w$, and does so in some finite number $t$ of steps. Since there exist inputs $x$ of length $t$ or more, on those inputs, $P(x)$ sees the acceptance and so takes the else branch of $S$. Thus $(P_4, S) \in IE$. Whereas:

$\langle M, w \rangle \notin A_{TM} \implies$ for all $x$, $P_4(x)$ does not see acceptance, no matter how long $x$ is and how long it simulates $M(w)$, so it always takes the then branch. Thus $(P_4, S) \notin IE$. 
All four of these answers suffice to show that the language $IE$ is undecidable. (So would the very-trivial answer $P_5 = \text{"if (M does not accept w) reject; else accept"} — but the note added to the question deemed it a violation of what an if-else statement means, at least when doing flowcharts. It is like answer $P_2$, anyway.) The differences show up in the continuation problem.

(2) With reference to Problem (1), suppose we are given a program $P$ such that we know in advance that a particular one of the following conditions holds. For each of the conditions, say whether knowing about it in advance and being able to assume it enables giving a definite yes/no answer for the If-Else problem about that particular $P$.

(a) For all inputs $x$, the computation reaches the statement $S$ at least once.

(b) For all inputs $x$, the computation reaches the statement $S$ exactly once, whereupon both the “yes” branch and the “no” branch halt after one more statement.

(c) On input $x = 10010110$ (150 in binary), the computation reaches the statement $S$ at least once, and we only need to decide whether that computation ever takes the “else” branch.

(d) On input $x = 10010110$ (150 in binary), the computation reaches the statement $S$ exactly once, and we only need to decide whether that computation takes the “else” branch.

In each case, if the answer is no, then you will be able to do the reduction in problem (1) in a way that the mapped programs $P = P_{M,w}$ also always satisfy the extra condition — regardless of whether $M$ accepts $w$ or not. (Perhaps you have already done so.) If you say yes to any condition, justify why. (24 pts. total)

**Answer:** With reference to $P_1, P_2, P_3, P_4$ from the previous problem:

(a) This extra condition holds for programs of the kind $P_3$ and $P_4$. The corresponding reductions guarantee this condition in both the $\langle M, w \rangle \in A_{TM}$ and $\langle M, w \rangle \notin A_{TM}$ instances, so even if you know it is in force, the question in the $IE$ problem is still undecidable.

(b) This extra condition is guaranteed by programs of the form $P_4$, again whether $\langle M, w \rangle \in A_{TM}$ or not. So even if you are **promised** that it holds, the question remains undecidable.

(c) This problem remains undecidable for programs of the form $P_3$, because they still have the “all-or-nothing” property with regard to $x$, with $x$ being ignored. (For programs of the form $P_4$, however, the extra condition means you only have to simulate $M(w)$ for 150 steps (or for 8 steps if you use the length), and a decision procedure can do that.)

(d) Regardless of the form of $P = f(M, w)$ with a designated statement $S$, if we are guaranteed in advance that the computation $P(10010110)$ reaches the statement $S$, and that we only have to observe which branch of $S$ is taken then-and-there, then we can “just do it.” We just run $P(x)$ with the advance notice that eventually it reaches $S$ where our analyzer (which you can picture as a debugger with a checkpoint on the statement $S$) can see what happens at that step. So the condition makes the problem decidable.
(3) Prove via the “all-or-nothing switch” that the following two problems are undecidable:

PAL1
INSTANCE: A deterministic Turing machine with input alphabet \( \Sigma = \{0, 1\} \).
QUESTION: Is there a palindrome \( x \) such that \( M \) accepts \( x \)?

PAL2
INSTANCE: A deterministic Turing machine with input alphabet \( \Sigma = \{0, 1\} \).
QUESTION: Does \( M \) accept all palindromes?

Also say which of these languages is c.e., if any. (24 pts. total)

Answer: For undecidability, we skill both birds with one tone—and we don’t even need to think about what a palindrome is. Just as in lecture, define \( f(M, w) = M' \) which on any input \( x \) runs \( M(w) \) open-endedly, and accepts \( x \) if and when \( M \) accepts \( w \). Then:

- \( \langle M, w \rangle \in A_{TM} \implies \text{for all } x, M' \text{ accepts } x \implies L(M') = \Sigma^* \implies M' \text{ accepts all palindromes} \implies f(M, w) = \langle M' \rangle \text{ is in both PAL1 and PAL2.} \)
- \( \langle M, w \rangle \notin A_{TM} \implies \text{for all } x, M' \text{ never gets to accept } x \implies L(M') = \emptyset \implies M' \text{ accepts no palindromes} \implies \langle M' \rangle \text{ is in neither PAL1 nor PAL2.} \)

Hence the switch simultaneously reduces \( A_{TM} \) to PAL1 and to PAL2, so both are undecidable. But to determine which one is c.e., we need a more particular proof. PAL1 is c.e., because its definition is completely existential over decidable predicates:

\[ \langle M \rangle \in \text{PAL1} \iff (\exists x, \tilde{x})[T(M, x, \tilde{c}) \land \tilde{x} \text{is a palindrome}]. \]

But PAL2 is not c.e., because of how its definition involves “all.” To show this in an ironclad way, we need to reduce a language we already know is not c.e. to it. Try the “delay switch” reduction with acceptance as the normal action and rejection as the “panic” action. In pseudocode this \( M' \) becomes like \( P_1 \) above but with the then/else branches switched (and the inessential stylistic change of using \( D_{TM} \) rather than the complement of \( A_{TM} \) as the source problem, so \( f(\langle M \rangle) = \langle M' \rangle \) is the reduction):

1. On input \( x \), set \( n = |x| \).
2. Do up to \( n \) steps of the computation \( M(\langle M \rangle) \).
3. if (\( M \) did not accept its own code in that time) accept \( x \); else reject \( x \);

The analysis is that if \( \langle M \rangle \in D_{TM}, \) i.e., if \( M \) does not accept its own code, then \( M' \) accepts all strings and hence accepts all palindromes, so \( f(\langle M \rangle) = \langle M' \rangle \) is in PAL2. But if \( \langle M \rangle \notin D_{TM} \) then \( M \) does accept its own code, so the “panic” condition happens at some time \( t \) and is seen for all strings \( x \) of length \( t \) or more. Then \( M' \) accepts none of those strings, so there are lots of palindromes that it fails to accept, so \( \langle M' \rangle \notin \text{PAL2} \). Thus \( D_{TM} \leq_m \text{PAL2} \), so PAL2 is not c.e. either. So it is neither c.e. nor co-c.e.
Recall that PAL stands for the language of palindromes over the alphabet $\Sigma = \{0, 1\}$. Prove that the language

$$\text{PAL}_3 = \{\langle M \rangle : \text{PAL} = L(M)\}$$

is neither c.e. nor co-c.e. (24 pts.) Then answer: What about the language

$$\text{PAL}_4 = \{\langle M \rangle : \text{PAL} \subseteq L(M)\}?$$

Is it c.e.? co-c.e.? neither-nor? (12 pts., for 36 total on the problem and 102 pts. on the written part of the problem set)

Answer: (a) This problem was quickly answerable if you read into the Wednesday 10/21 lecture. Let PAL stand for the language of palindromes themselves—it was important not to confuse this with any of PAL1 thru PAL4 which are all languages whose constituents are (codes of) machines. Do the same reduction and observe that the “yes” case makes $L(M') = \text{PAL}$ as well as making $L(M')$ non-regular, whereas the “no” case makes $L(M') = \emptyset$ so $L(M') \neq \text{PAL}$. Do we get a reduction from $\text{A}_{TM}$ to PAL3 or from the complement of $\text{A}_{TM}$ to PAL3? It is from $\text{A}_{TM}$. So PAL3 is not co-c.e. (You can also get this from Rice’s Theorem by noting that PAL3 is the index set of the class $\{\text{PAL}\}$, whose only member is PAL, and this class does not include the empty language, so you get that the index set is not co-c.e.)

We still need to prove that PAL3 is not c.e. We can re-use the proof given in the latter part of problem (3) but make the accepting branch filter for palindromes. That is, make $g(M) = \langle M'' \rangle$ where $M''$ is a TM that executes the routine:

1. On input $x$, set $n = |x|$.
2. Do up to $n$ steps of the computation $M(\langle M \rangle)$.
3. if (M did not accept its own code in that time) accept $x$ only if $x$ is a palindrome; else reject $x$;

Then $g$ reduces $\text{D}_{TM}$ to PAL3, so PAL3 is not c.e. either. So it is neither c.e. nor co-c.e. (It is in fact mapping-equivalent to $\text{ALL}_{TM}$ and to TOT, and proving that was another way to answer the question—with just one reduction.)

(b) This is the same language as PAL2, since the condition $\text{PAL} \subseteq L(M)$ is the same as saying that $M$ accepts all palindromes. Hence the above proof for the latter part of problem (3) answers this question. [Thus, if you read that part as a bare yes/no question not requiring proof, the proof came here anyway.]