(1) Show that the concatenation of any two decidable languages is decidable, and that the concatenation of any two c.e. languages is c.e. Find, however, an example of two undecidable languages $A$ and $B$ such that $A \cdot B$ is decidable. Indeed, you can arrange that $A \cdot B$ is regular and even make $B = A$. (Vague hint: Take any language $L \subset \{0,1\}^*$ whatever and “dilate it” in such a way that you can fill in lots of extra strings into the “gaps” you made but the resulting language $A$ is still undecidable. $9+6+12 = 27$ pts.)

Answer: (a) Given decidable languages $A$ and $B$, we may take total Turing machines $M_a$ and $M_b$ deciding them, and on any input $x \in \Sigma^*$ run the following loop:

\[
\begin{align*}
    n & := |x|; \\
    \text{for } i = 0 \text{ to } n \text{ do} & \\
        \quad \text{string } y = x[1..i], z = x[i+1..n]; \quad \text{//note: indexing from 1} \\
        \quad \text{if } (M_a \text{ accepts } y \&\& M_b \text{ accepts } z) \{ \\
            \quad \quad \text{accept } x; \\
        \quad \} \\
\end{align*}
\]

Since $M_a$ and $M_b$ are total the body of this loop always terminates, and since the loop is strictly counted, the loop always finished too. Thus this routine always halts, and it accepts iff $x \in A \cdot B$, so $A \cdot B$ is decidable.

(b) If we only know that $A$ and $B$ are c.e. then we cannot code a machine for $A \cdot B$ serially as above, because the failure of one loop iteration to halt will prevent a later iteration from ever getting the chance to succeed. Instead what we need to do is fork $n+1$ processes in parallel and accept $x$ if and when one of them gets notified of acceptance of both $M_a(y)$ and $M_b(z)$.

An alternate answer is to insert a master governing while loop:

\[
\begin{align*}
    n & := |x|; \\
    t & = 0; \\
    \text{while (true) } \{ \\
        t++; \\
        \quad \text{for } i = 0 \text{ to } n \text{ do} \\
            \quad \quad \text{string } y = x[1..i], z = x[i+1..n]; \quad \text{//note: indexing from 1} \\
            \quad \quad \text{Run } M_a(y) \text{ and } M_b(z) \text{ for } t \text{ steps each; } \\
            \quad \quad \text{if } (\text{both accepted within that time}) \{ \\
                \quad \quad \quad \text{accept } x; \quad \}
        \}
\end{align*}
\]

This is like the proof of $\text{REC} = \text{RE} \cap \text{coRE}$ given in class.

(c) Take $C$ to be any undecidable language. Then its complement $\bar{C}$ is undecidable too. So are $A = C \cup \{\lambda\}$ and $B = \bar{C} \cup \{\lambda\}$ since we’re adding at most one string to one of them. But $A \cdot B = \Sigma^*$ which is decidable, regular, everything.

To do this with $B = A$ needs “dilating” $C$ further as vaguely hinted. Take $O$ to be the set of odd-length strings over $\Sigma$. Define $C' = \{xx : x \in C\}$. Then $C'$ is likewise undecidable and
every string in $C'$ has even length. Finally define

$$A = \{\lambda\} \cup O \cup C'.$$

We still have for all $x$ (other than $\lambda$) that $x \in C \iff xx \in A$ so $A$ remains undecidable. But $A \cdot A$ includes all the odd-length strings via $\lambda \cdot O$, all the even-length strings other than $\lambda$ since they can be broken into two odd-length strings and so belong to $O \cdot O$, and includes $\lambda$ via the rule $\lambda \cdot \lambda = \lambda$. So $A \cdot A = \Sigma^*$ which is decidable, even regular.

(2) Prove—preferably by reduction—that the following decision problem is undecidable:

**Instance:** A Turing machine $M$.

**Question:** Is every string accepted by $M$ a palindrome?

State the language $L$ of this problem, then state the complementary language $\tilde{L}$ by a prose definition, and finally sketch why $\tilde{L}$—though likewise undecidable—is computably enumerable.

**Answer:** Take $P$ to be the language of palindromes. Note that $P$ is decidable—it is in deterministic linear time and also has an NPDA but not a DPDA (asserted but not proven in class—the proof is hard) and certainly not a DFA (proved by MNT). One should not confuse $P$ with $L$—instead we use $P$ to define $L$:

- $L = \{M : L(M) \subseteq P\}$.
- $\tilde{L} = \{M : (\exists x)x \in L(M)$ but $x \notin P\}$.

The “all-or-nothing-switch” as given in the Monday 10/10 class solves this literally: Map an instance $\langle M, x \rangle$ of the $A_{TM}$ problem to a machine $M'$ that on any input $w$ runs $M(x)$, and accepts $w$ only if and when $M$ accepts $x$. Then

$$\langle M, x \rangle \in A_{TM} \implies L(M') = \Sigma^*,$$

which certainly means that $M'$ accepts some non-palindromes. So $M' \in \tilde{L}$. But

$$\langle M, x \rangle \notin A_{TM} \implies L(M') = \emptyset.$$

Now this may sound like sophistry, but when $L(M') = \emptyset$ it is always considered true (“by default”) that every string accepted by $M'$ is a palindrome, likewise that every string accepted by $M'$ appears in the *Dao De Jing*.\(^1\) So this is a valid answer and makes $M' \in L$. So

$$\langle M, x \rangle \notin A_{TM} \iff M' \in \tilde{L},$$

which means that $A_{TM}$ is many-one reducing to the complement of $L$. Hence $L$ is undecidable.

It is important to realize right away that this also rules out $L$ being c.e., but the final part of the question asked about $\tilde{L}$. This is c.e., and here’s why: We can run a master loop for $n = 1, 2, 3, 4, \ldots$, and on each iteration, run $M$ on the first $n$ strings that are not palindromes, for $n$ steps on each such string. If $M \in \tilde{L}$, then there is some non-palindrome $x$ that it accepts, and we will catch this whenever $n$ is both bigger than the numerical value of $x$ and than the number of steps it took to accept $x$. If not, then our loop never catches such an $x$ and hence never accepts. Thus the loop represents a Turing machine $T$ such that $L(T) = \tilde{L}$, so $\tilde{L}$ is c.e., so $L$ is co-c.e.

\(^1\)Verse 11: “We mold clay into a pot, but it is the emptiness inside that makes the vessel useful.”