1. **(25 points):** Kozen HW 9, Question 3
   Prove that the emptiness problem for linear bounded automata is undecidable.

   Solution: I will show that VALCOMPS can be solved by a LBA. As shown in Kozen, lecture 35 (eq. 35.1),

   $$\text{VALCOMPS}(M, x) = \emptyset \quad \Leftrightarrow \quad M \text{ does not halt on } x.$$

   Let $M_0 = (Q_0, \Sigma_0, \Gamma_0, \vdash_0, \square_0, \delta_0, s_0, t_0, r_0)$ be a Turing machine. Define an LBA $M = (Q, \Sigma, \Gamma, \vdash, \dashv, \delta, s, t, r)$ with

   $$\begin{aligned} \Sigma &= \Gamma_0 \times (Q_0 \cup \{\circ\}) \\ \Gamma &= \{\vdash, \dashv, \#\} \cup \Sigma \cup ((\Sigma \cup \{\#\}) \times \{[]'\}) \end{aligned}$$

   In other words, $\vdash$ and $\dashv$ are the left and right endmarkers for M. The symbol (c, q) indicates that the symbol on the tape for M is c, the read/write head for M_0 is at this square, and M_0 is in state q. The symbol $(c, \circ)$ indicates that the symbol on the tape for M_0 is c and the read/write head for M_0 is not at this square. Tape symbols for M can be marked with a $'$ which I will use below. The tape symbol $\#$ is used to separate successive configurations of M.

   Rather than listing out all of the states and the details of the state transition relation, I will describe the operations of M, in each case making it clear that they could be performed by an LBA.

   - Initially, M reads its input tape an makes sure that it has the right structure to describe a valid computation. In particular, in makes sure that the input tape has the form $\#w_0\#w_1\#w_2\#\ldots\#w_k\#$ where each w_i is a string in $(\Gamma_0 \times (Q_0 \cup \{\circ\}))^*$, each w_i contains exactly one symbol in $\Gamma_0 \times \{\circ\}$, w_0 encodes the initial configuration for M_0 with input x, and w_k encodes a configuration for M_0 in an accepting or rejecting state. As chapter 35 of Kozen, these restrictions are a regular language. The LBA has a finite state control. Thus it can test these conditions on a single pass over the tape. If the tape fails any of these conditions the LBA rejects. Otherwise, it returns to the left endmarker and continues with the steps described below.
   - M now marks the leftmost $\#$ and the first $\#$ to the right of the leftmost one with $'$s. M then returns to the left endmarker.
   - M now checks that successive configurations as described on its input tape correspond to legal moves of M_0. M maintains its tape with two squares marked with $'$s; these mark corresponding tape locations of two consecutive positions. M finds the left marked square and remembers the tape symbol for M_0 in M's own finite state. It moves one square to the right.
     If this square has the tape head marker for M_0, then M memorizes the tape symbol and state for M_0 symbol along with the tape symbols of M_0 for the previous and following square in M's own finite store. M then moves to the right to find the marked square in the next configuration and checks that that the three tape symbols starting with the marked one correspond to the successor by δ_0 of the three that it has memorized in its finite store. If not, M rejects. Otherwise, M moves the $'$ markers for both configurations to the squares immediately after the groups of three. M also remembers if the state of M_0 in the successor configuration is a final state (i.e. t_0 or r_0).
     Otherwise, the square after the marked one does not encode the tape head marker for M_0. In this case, M just scans to the marked square in the successor configuration and confirms that it matches the symbol from the predecessor configuration. If not, M rejects. Otherwise, M moves the $'$ marks one to the right for each configuration.
     If M reaches its own right endmarker, $\dashv$, M recalls from its finite store whether or not M_0 was in a terminal state in the final configuration. If so, M accepts, otherwise, M rejects.

   This machine accepts a string, iff it describes an terminating computation of M on input x. If M_0 terminates on input x, the corresponding computation history is accepted by M. Otherwise, the language accepted by M is empty. The halting problem for Turing machines is undecidable. Thus, the language emptiness problem for LBAs is undecidable as well.

2. **(25 points):** Kozen HW 10, Question 1
Show that neither the set

$$\mathsf{TOTAL} \stackrel{\text{def}}{=} \{M \mid M \text{ halts on all inputs}\}$$

nor its complement is r.e.

Solution: I start from the observation that the halting problem is r.e. but is not co-r.e. Let M be a TM and x be an input to M. I'll now construct a new TM, M' such that on input y, M' simulates M running on input x for $|y|$ steps. If M does halt within $|y|$ steps, then M' will go into an infinite loop. Otherwise, M' will accept y. By this construction, M' is total iff M does not halt on input x. This reduces the complement of the halting problem, $\sim HP$, to TOTAL. We know that $\sim HP$ is not r.e. Thus, TOTAL in not r.e. either.

A similar construction works to show that TOTAL is not co-r.e. This time, we make a TM M' when run with input y first simulates M running on input x. If M halts, then M' accepts y (whatever y happens to be). Otherwise, M' simulates M forever. By this construction, M' is total iff M halts on input x. This reduces the halting problem, HP, to TOTAL. We know that HP is r.e. but not co-r.e. Thus, TOTAL in not co-r.e. either.

Note: I first tried to solve this using Rice's theorem, but Rice's theorem applies to the language recognized by the TM, not to the TM itself. Thus, I gave up on that approach and looked for a reduction argument instead.

3. **(25 points):** Kozen HW 10, Question 3
Show that it is undecidable whether the intersection of two CFLs is non-empty.

Solution: As suggested by the hint in the textbook, I'll turn this into a variation of VALCOMPS. Given a TM M with input string X, I'll ask whether $\#\alpha_0\#\alpha_1^R\#\alpha_2\#\alpha_3^R\#\ldots\#\alpha_n$ encodes a valid configuration, where α_i is gives the configuration in reverse when i is odd in in normal order when i is even.

The basic idea is to use two PDAs. The first PDA checks that each even numbered configuration is followed by the correct configuration. It does this by pushing the even numbered configuration onto the stack and then popping each symbol off while checking the symbols of the subsequent odd numbered configuration. Thus, this PDA checks that α_1^R is the valid successor to α_0, that α_3^R is the valid successor to α_2 and so on. The second PDA checks that each odd numbered configuration is followed by the correct even numbered configuration. In other words, it verifies that α_2 is the valid successor to α_1^R and so on. If both PDAs accept and the final configuration is a final configuration for M, then the input string encoded a valid computation of M on input x.

I'll assume that the input string has the valid structure for VALCOMPS (i.e. that each configuration has exactly one symbol that marks the state of M in that configuration, etc.). As pointed out in my solution to problem 1, this is a regular language. It can be incorporated into either or both PDAs.

Now, I'll describe the operation of the first PDA. It reads each symbol in α_{2i}. If the symbol does not include the marker for the read/write head of M, the PDA just pushes the symbol onto its stack. If it does include the marker for the read/write head, then the PDA memorizes the value of the previous symbol (on the top of the stack), the current symbol, and the state for M in its finite state, and reads the next symbol. The PDA can now simulate the move of M and pushes the appropriate symbols onto its stack. If the PDA determines that M enters a final state, the PDA remembers this in its own state, and all states of the PDA from here on are accepting states (unless the PDA discovers an error elsewhere in the configuration).

When the PDA reaches the $\#$ symbol, the stack holds the next configuration of M. The PDA reads α_{2i+1}^R and pops these symbols off of the stack as it goes, comparing the symbol it reads with the symbol on the top of the stack at each move. If they agree, the PDA continues, otherwise, it enters into a permanently rejecting state. If the PDA pops all of the symbols off of its stack in this maneer, then α_{2i+1}^R is the valid successor of α_{2i}. The PDA verifies that the next input symbol is a $\#$ and then repeats this whole process (to verify that α_{2i+2} is properly followed by α_{2i+3}.

The operation of the second PDA is similar.

If both PDAs accept, then $\#\alpha_0\#\alpha_1^R\#\alpha_2\#\alpha_3^R\#\ldots\#\alpha_n$ encodes a valid computation. The language accepted by a PDA is a CFL. Thus, the language accepted by both of these PDAs is the conjunction of two CFLs. This language is non-empty iff M halts on input x. The halting problem is undecidable. Thus, the language non-emptiness problem for the intersection of two CFLs is undecidable.

Note: my construction made no use of non-determinism. Thus, the language non-emptiness problem for the intersection of two DCFLs is undecidable as well.

4. **(25 points):** Kozen Miscellaneous Exercises, Question 103.
A nondeterministic Turing machine is on ewith a multiple-valued transition relation. Give a formal definition of these machines. Argue that every nondeterministic TM can be simulated by a deterministic TM.

Solution: A non-deterministic TM is a 9-tuple, $M = (Q, \Sigma, \Gamma, \vdash, \square, \Delta, s, t, r)$, where Q is a set of states; Γ is a finite tape alphabet; $\Sigma \subset \Gamma$ is a finite input alphabet; $\vdash$ is the tape left-endmarker; $\square \in \Gamma$ is the blank symbol; $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ is the state transition relation; and s, t, and r are the start, accept, and reject states respectively. In particular, $((q, c), (q', c', d)) \in \Delta$ means that when M is in state q reading symbol c, M can enter state q', write c' on the tape, and move the read/write head one square in direction d. For any particular q and c, there may be multiple choices for q', c', and d.

To simulate a non-deterministic TM, M_{nd}, with a deterministic TM, M_d, I'll use the usual universal machine construction (as in Kozen, chapter 31). However, I'll add a symbol, $\#$ to the tape alphabet of M_d. and keep track of each possible configuration of M_{nd} on the tape of M_d. Based on the construction from Kozen, chapter 31, M_d will have three tracks, with the top track holding the description of M_{nd}, the middle track holding the contents of the tape for M_{nd} for each possible configuration, and the bottom track marking the tape head position and state of M_{nd}, again for each possible configuration. Thus, the middle track will look like:

$$w_1\#w_2\#\ldots\#w_k$$

if there are currently k configurations of M_{nd} being simulated.

For each simulation step, M_d will simulate each configuration of M_{nd} for one step. Let w_i denote such a configuration. If from w_i, M_{nd} can move to an accepting configuration, then M_d accepts. If w_i has m possible successors, then M_d will make $m-1$ copies of w_i at the end of its tape and update the original one and the $m-1$ copies to reflect each possible move of m. If w_i has no successors, or if all successors enter the rejecting state, then M_d erases w_i from its tape. If a successor of w_i moves beyond the $\#$ at the end of w_i, then M_d shifts all symbols to the right of w_i one tape square to the right and appends a bland ot w_i.

Machine M_d accepts if there is any sequence of choices for M_{nd} such that M_{nd} accepts. Thus, M_d simulates M_{nd}.

Note that it was crucial for M_d to simulate each possible configuration of M_{nd} at each step. In particular, M_d can't just simulate one of the non-deterministic choices until it accepts or rejects and then consider the others if the first choice rejects. This is because some choices may lead to looping, while a different choice would accept. If M_d followed the looping choice without considering the others, M_d would fail to terminate, even though M_{nd} would accept.