5.4 No. For example, consider \( A = \{0^n1^n \mid n = 0, 1, 2, 3, \ldots \} \) and let \( f \) be the function that maps \( 0^n1^n \) to \( 1^n \) and maps all other string to -1. Then \( f \) is a mapping reduction from \( A \) to \( 1^* \), which is regular even though \( A \) is not regular.

5.5 Show that \( A_{TM} \) is not mapping reducible to \( E_{TM} \). I will be using the \( \neg \) symbol to indicate complement instead of drawing a line over the language. We know that \( A_{TM} \) is Turing recognizable, but not co-Turing recognizable. The TM below recognizes \( \neg E_{TM} \), so \( E_{TM} \) is co-Turing recognizable.

\[
M = \text{"On input } <M> \text{, where } M \text{ is a TM}
1. \text{For each } i = 1, 2, 3, \ldots
2. \text{Run } M \text{ on all strings of length } i \text{ for } i \text{ steps}
3. \text{If any string is accepted, accept"}
\]
This Turing machine will accept any Turing machine whose language is non-empty.

Now assume \( A_{TM} \) is mapping reducible to \( E_{TM} \). Then \( \neg A_{TM} \) is mapping reducible to \( \neg E_{TM} \). But, \( \neg E_{TM} \) is Turing recognizable and \( \neg A_{TM} \) is not, which contradicts Theorem 5.22. This is a contradiction. Therefore, \( A_{TM} \) is not mapping reducible to \( E_{TM} \).

5.7 \( A \leq_m \neg A \) implies \( \neg A \leq_m A \). By Theorem 5.16, we can conclude that \( \neg A \) is Turing recognizable since we know \( A \) is Turing recognizable. By Theorem 4.16, we can conclude \( A \) is decidable since it is both TR and co-TR.

5.3 Let \( A \) be any Turing recognizable language and let \( M \) be a Turing machine such that \( L(M) = A \). Let \( f \) be the function that maps any string \( w \) to the string \( <M.w> \). Then \( w \) is in \( A \) if and only if \( f(w) \) is in \( A_{TM} \) – i.e., \( f \) is a mapping reduction from \( A \) to \( A_{TM} \).

6.3 Since \( A \leq_T B \), there is a Turing machine \( M_1 \) that calls an oracle for \( B \) and decides \( A \). Similarly, since \( B \leq_T C \), there is a Turing machine \( M_2 \) that calls an oracle for \( C \) and decides \( B \). Now, consider the Turing machine \( M \) that does exactly what \( M_1 \) does except instead of calling the oracle for \( B \), it calls \( M_2 \). Since \( M_2 \) decides \( B \), it will give the same answer as the oracle did, so \( M \) will decide \( A \). Also, \( M_2 \) uses an oracle for \( C \), so \( M \) also uses and oracle for \( C \) to decide \( A \). Therefore, \( A \leq_T C \).