12.10 Let \( M \) be a T.M. such that \( L(M) \) is not recursive. Suppose that the
decision problem: does \( M \) accept \( w \) is decidable. Then, given \( \Sigma \), we can generate
\( \Sigma^* = \{w_0, w_1, \ldots\} \) in canonical order. By successively asking: does \( M \) accept \( w_i \)
and incrementing \( w_i \) we can determine every word in \( L(M) \) for \( i < N \) for any given
\( N \). But since \( \Sigma^* \) is in canonical order, this generates \( L(M) \) in canonical order, by
incrementing \( N \). But if we can generate \( L \) in canonical order, then it is recursive
by theorem 10.6. Therefore the assumption leads to contradiction.

Many alternative proofs - another approach is to show that solving the decision
problem for a non-recursive language allows us to decide the halting problem, which
we know is undecidable.

12.15 a) Given a C program \( P \) and a specific statement \( s \), is \( s \) executed on input
data \( I \). Reduce the C program to a T.M as follows: let each statement \( s \) be a set of
TM states \( Q_s = \{q|q \in s\} \) of \( M \), which get executed in some sequence, with memory
access being reads/writes to tape. Let the memory stand for the tape, where each
memory address is the number of a particular cell in the tape (reads and writes
to non-adjacent memory addresses imply the machine can move a fixed number
of steps from one address to another, but we have seen T.M.s that can do this, and
we call them as subroutines). Now construct a machine \( M_s \) by replacing a T.M.
state in the set \( Q_s \) by a halt instruction. Then the question does \( s \) get executed by
\( P \) with input \( I \) is equivalent to: does \( M_s \) halt with input \( I \), which is the halting
problem and is known to be unsolvable.

Different constructions are possible.

12.15 b) Same construction as above. But now the problem is equivalent to
does \( M_s \) halt on any input \( I \). This is equivalent to: is the language accepted by \( M_s \)
empty, which is unsolvable. Different constructions are possible.

12.17 Given T.M., \( M \), examine the states of the machine that read a blank
and write a blank. If the start state is not included in a cycle of such states, then
the machine must write a non-blank. Key to this is examining the structure of the
particular machine, rather than its output.

12.21 PCP with two symbols is equivalent to PCP with \( N \) symbols because we
can group symbols into substrings of fixed length, thereby creating more symbols
(for example, with substrings 8 characters long we get the 8 bit ASCII codes). If
we could solve PCP for an alphabet of two symbols, then we could solve it for these
larger groups and this would solve general PCP.