3.8 b. "On input string w:
1. Scan the tape and mark the first 0 which has not been marked.
   If there is no unmarked 0, go to stage 5.
2. Continue scanning and mark the next unmarked 0. If there is
   not any on the tape, reject. Otherwise, move the head to the
   front of the tape.
3. Scan the tape and mark the first 1 which has not been marked.
   If there is no unmarked 1, reject.
4. Move the head to the front of the tape and repeat stage 1.
5. Move the head to the front of the tape. Scan the tape for any
   unmarked 1s. If none, accept. Otherwise, reject."

c. "On input string w:
1. Scan the tape and mark the first 0 which has not been marked.
   If there is no unmarked 0, go to stage 5.
2. Continue scanning and mark the next unmarked 0. If there is
   not any on the tape, accept. Otherwise, move the head to the
   front of the tape.
3. Scan the tape and mark the first 1 which has not been marked.
   If there is no unmarked 1, accept.
4. Move the head to the front of the tape and repeat stage 1.
5. Move the head to the front of the tape. Scan the tape for any
   unmarked 1s. If none, reject. Otherwise, accept."

3.12 We simulate an ordinary TM with a reset TM that has only the RESET and
R operations. When the ordinary TM moves its head right, the reset TM
does the same. When the ordinary TM moves its head left, the reset TM
cannot, so it gets the same effect by marking the current head location on
the tape, then resetting and copying the entire tape one cell to the right,
except for the mark, which is kept on the same tape cell. Then it resets
again, and scans right until it finds the mark.

c. For any decidable language $L$, let $M$ be the TM that decides it. We construct
a NTM $M'$ that decides the star of $L$:
   "On input w:
   1. For each way to cut w into parts so that $w = w_1 w_2 \ldots w_n$:
   2. Run $M$ on $w_i$ for $i = 1, 2, \ldots, n$.
       If $M$ accepts each of these string $w_i$, accept.
   3. All cuts have been tried without success, so reject."
If there is a way to cut w into different substrings such that every substring is
accepted by $M$, w belongs to the star of $L$ and thus $M'$ accepts w. Otherwise,
w is rejected. Since there are finitely many possible cuts of w, $M'$ will halt
after finitely many steps.

e. For any two decidable languages $L_1$ and $L_2$, let $M_1$ and $M_2$ be the TMs that
decide them. We construct a TM $M'$ that decides the intersection of $L_1$ and
$L_2$:
   "On input w:
   1. Run $M_1$ on w, if it rejects, reject.
   2. Run $M_2$ on w, if it accepts, accept. Otherwise, reject."
$M'$ accepts w if both $M_1$ and $M_2$ accept it. If either of them rejects, $M'$
rejects w, too.
3.16 b. For any two Turing-recognizable languages \( L_1 \) and \( L_2 \), let \( M_1 \) and \( M_2 \) be the TMs that recognize them. We construct a NTM \( M' \) that recognizes the concatenation of \( L_1 \) and \( L_2 \):

"On input \( w \):

1. Nondeterministically cut \( w \) into two parts \( w = w_1w_2 \).
2. Run \( M_1 \) on \( w_1 \). If it halts and rejects, reject.
3. Run \( M_2 \) on \( w_2 \). If it accepts, accept. If it halts and rejects, reject."

If there is a way to cut \( w \) into two substrings such \( M_1 \) accepts the first part and \( M_2 \) accepts the second part, \( w \) belongs to the concatenation of \( L_1 \) and \( L_2 \) and \( M' \) will accept \( w \) after a finite number of steps.

3.18 If \( A \) is decidable, the enumerator operates by generating the strings in lexicographic order and testing each in turn for membership in \( A \) using the decider. Those strings which are found to be in \( A \) are printed.

If \( A \) is enumerable in lexicographic order, we consider two cases. If \( A \) is finite, it is decidable because all finite languages are decidable. If \( A \) is infinite, a decider for \( A \) operates as follows. On receiving input \( w \), the decider enumerates all strings in \( A \) in order until some string appears which is lexicographically after \( w \). That must occur eventually because \( A \) is infinite. If \( w \) has appeared in the enumeration already then accept, but if it hasn't appeared yet, it never will, so reject.

Note: Breaking into two cases is necessary to handle the possibility that the enumerator may loop without producing additional output when it is enumerating a finite language. As a result, we end up showing that the language is decidable but we do not (and cannot) algorithmically construct the decider for the language from the enumerator for the language. This subtle point is the reason for the star on the problem.