3.2 b  The simulating deterministic decider D has three tapes. By Theorem 3.8 this arrangement is equivalent to having a single tape. The machine D stores the input on the first tape – this tape is never changes. The second tape maintains a copy of N’s tape on some branch of its nondeterministic computation. Tape 3 keeps track of D’s location in N’s nondeterministic computation tree. **Since N is a decider, every branch of this tree ends in a leaf – either in the accept state or in the reject state.** Let P be the set of all possible paths from the root of this tree to some leaf.

Here is the description of D:

1. Initially, tape 1 contains the input w and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Write a path from P on tape 3.
4. Use tape 2 to simulate N with input 2. At each decision point, use tape 3 to determine which choice to make.
5. If this branch ends in an accept state, accept.
6. Otherwise, this branch ends in a reject state. **Check if all possible branches in P have been tried. If so, reject.**
7. Otherwise, the tree has not yet been exhaustively searched. Replace the string on tape 3 with the next element in P. Simulate the next branch of N’s computation by going to stage 2.
3.8 a

1. Using a 2-tape machine, start with w on tape 1 and tape 2 empty.
2. Write a Y on tape 2.
3. Read the first symbol on tape 1.
   a. If it’s a 0, change it to an X, write an X on tape 2, and move both tape heads right.
   b. If it’s a 1, change it to a Y and move tape 1’s tape head right.
   c. If it’s a ~, accept.
4. Repeat until the tape 1 head reads a ~:
   a. If tape 1’s head is on a 0, write an X on tape 2 and move both tape heads right.
   b. If tape 1’s head is on a 1, move tape 1’s head right.
5. (At this point, there is one X on tape 2 for each 0 in w). Move left on tape 1 until the tape head encounters an X or Y.
6. Move left on tape 2 until the tape head encounters a Y. Move right one space on tape 2.
7. If tape 1’s head is on a Y, change the Y to a 1 an move tape 1’s head to the left.
8. Repeat until either tape head encounters a ~:
   a. Move tape 1’s head right until a 1 is encountered.
   b. Move both tape head’s right one space.
9. If tape 1 has encountered a ~ and tape 2 is on an X, reject.
10. Move tape 1’s head right until the first 1 or ~
    a. If a 1 is encountered, reject.
    b. Otherwise, accept.
Consider the following 2-PDA. The notation $x,(a_1,a_2) \rightarrow (b_1,b_2)$ indicates the symbol $x$ is read and stack 1’s head is changed from $a_1$ to $b_1$, and stack 2’s head is changed from $a_2$ to $b_2$.

This 2-PDA accepts the language $a^n b^n c^n$, which we know is not a context-free language. Clearly, the class of 2-PDA’s is at least as powerful as 1-PDA’s, since any language accepted by a 1-PDA can be accepted by a 2-PDA – simply have the 2-PDA perform the same actions as the 1-PDA and use only one stack. Therefore, the class of 2-PDA’s is strictly more powerful than the class of 1-PDA’s.
3.9 b  Clearly a 3-PDA is at least as powerful as a 2-PDA (for the same reason that a 2-PDA is at least as powerful as a 1-PDA). Also, a 3-PDA is no more powerful than a 3-tape TM, since any 3-PDA could be simulated by a 3-tape TM. Therefore, if we can show that a 2-PDA is as powerful as a 3-tape TM, then it must be the case that 3-PDA’s and 2-PDA’s have the same power. Since 3-tape TM’s have the same power as 1-tape TM’s, it suffices to show that 2-PDA’s have the same power as 1-tape TM’s.

The configuration $uqv$ can be simulated on a 2-PDA by having $u$ on stack 1 with the beginning of $u$ on the bottom of the stack and the end of $u$ at the top of the stack, and $v$ on stack 2 with the beginning of $v$ on the top of the stack and the end of $v$ on the bottom of the stack. In addition, below the last character of $v$ on stack 2 is a $\sim$ to mark the end of the word.

The 2-PDA will have the same states as the TM. The Turing transitions can be simulated by carefully moving symbols from one stack to the other. If the TM has a transition where $uaq_i bv$ yields $uq_j acv$, and the 2-PDA is in state $q_i$ with $b$ on the top of stack 2, the 2-PDA should change the $b$ to a $c$, pop stack 1, push the popped symbol on to stack 2, and go to state $q_j$. Similarly, if $uaq_i bv$ yields $uacq_j v$, and the 2-PDA is in state $q_i$ with $b$ on the top of stack 2, the 2-PDA should pop the $b$ off stack 2, push a $c$ on to stack 1, and go to state $q_j$. 
3.12 It suffices to show that this type of TM can simulate a standard TM. If we add a tape to the TM (we can add a tape without adding power … this can be proved using the same techniques used in the proof of Theorem 3.8), that keeps track of where we are in the string, we can move left by resetting and moving right to the point just before where we were prior to the reset.

More specifically, this TM would simulate a standard TM by maintaining a “left position” tape in addition to the calculation tape. When the calculation tape head is at the beginning of the tape and moves right, nothing is done to the left position tape. When the calculation tape head is not at the beginning of the tape and moves right, the left position tape writes an X and moves right. This way, the left position tape points to the position to the left of the tape head. When the TM transition indicates a left move, reset both tapes and move them both right until you reach the end of the string on the left position tape. At this point the left position tape needs to be changed to reflect that the tape head has moved left. Do this by setting a marker on the calculation tape (e.g., an X if the tape head is on a 0 and a Y if the tape head is on a 1), reset both tapes, move right one on the calculation tape then move both tapes right in tandem until the marker is reached – at that point write a ~ on the left position tape. In this manner, any standard TM operations can be simulated. Therefore, these two types of machines have the same power.

3.13 If you can only move right on the tape, you essentially have no memory. Therefore, this is essentially a DFA … these machines recognize regular languages.