D) Note that the naming of certain symbols is completely arbitrary, so that part of my explanation can be discarded if it makes this answer too non-generalized; the naming scheme for symbols is based purely off the way the homework PDF does so.

To generate a CFG with sets N, upper-sigma, I, and R from an FSA, first generate the finite set of non-terminal symbols. We can generate N such that each element corresponds to:
- the entry point into the CFG (labeled S in the homework)
- a non-terminal symbol (labeled A in the homework) whose purpose is to prefix the output string with a start-marker in the rules
- a labeled state in the FSA (labeled “X#” in the homework)
- a concatenation transition in the FSA (in the homework, symbol name collisions are avoided by assigning these non-terminal symbols the upper-case equivalent of the corresponding transition’s lower-case symbol).

Again, note that these naming schemes are completely arbitrary, but overall, the naming scheme should try to maintain some level of correlation between states, transitions, and the associated non-terminal symbols while still avoiding collisions.

To generate upper-sigma, the finite list of terminal symbols, we can simply take the alphabet of the FSA and add the start and end markers to that set.

Set I can be assigned to the non-terminal symbol corresponding to an entry point into the FSA (labeled S on the homework).

To generate the set R of rules, follow the following process:

- There should be a rule such that the non-terminal symbol corresponding to the entry point into the FSA (‘S’) is substituted for the string of non-terminal symbols:
(rule to prefix the start marker onto the output) ++ (initial state of FSA)
This corresponds to the rule S → A X1 (or any other initial state in the FSA) in the examples given on the homework.
- There should be a rule to prefix the start marker onto the output string. This corresponds to rule A → {start} in the homework.
- There should be a rule such that the input non-terminal symbol corresponds to the origin point of a transition. The output string of non-terminal symbols is of format:
(non-terminal symbol corresponding to a transition) ++ (non-terminal symbol corresponding to target state of transition within FSA)
For example, the corresponding transition within an FSA from state 1 to state 2, concatenating a ‘c’ following the style shown in the homework would be X1 → C X2.
- There should a rule for each non-terminal symbol corresponding to a transition such that the input non-terminal symbol is substituted with a terminal symbol corresponding to the symbol concatenated by the transition. For example, C → c.

- Finally, there should be a rule such that the input non-terminal symbol is one that corresponds to an exit state within the FSA. This symbol is substituted for a terminal symbol for the end-marker of the output string. For example, an FSA that allows state 1 to be an exit point would generate the corresponding CFG rule X1 → {end}.

E)

	… {start}	… c	… v	… c	… {end}
{start} …	S: 0 A: 1 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 1 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0
c …		S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 1 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 1** C: 0 V: 0
v …			S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 1	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 A: 0 **X1: 1** **X2: 1** **X3: 0** C: 0 V: 0
c …				S: 0 A: 0 **X1: 0** **X2: 0** **X3: 0** C: 1 V: 0	S: 0 A: 0 **X1: 0** **X2: 0** **X3: 1** C: 0 V: 0
{end} …					S: 0 A: 0 **X1: 1** **X2: 0** **X3: 0** C: 0 V: 0

Note that I understood the non-diagonal entry (column 5, row 3) to mean “cc”, not “c” like its diagonal equivalents.

Note that the only values with any non-zero inside values for X1-X3

occur when the suffix is {end}. To suffix with {end}, a rule's final non-terminal symbol added needs to be X1. In an FSA, this is equivalent to having a transition ending at state 1. Now, note the character columns in the forward values table that have a forward value of 1 for state 1 on the character column's right-hand side. These correspond to (c)v(c), …(v)c, and {epsilon} (you can place this as a prefix or suffix for cvc and take the forward values leading into the first c). For these characters, now check the corresponding left-hand backwards values. These backwards values correspond perfectly to the inside values in the graph for the corresponding character preceding {end}. To put it more clearly, for characters that have a forward value of 1 for going into state 1 (since characters in the forward values graph are in between columns, consider these the "right hand" forward values), the corresponding characters in the backwards values graph have left-hand backwards values that are identical to their inside values when preceding {end}.

F) It seems like the rules in F) are of form:

(target) → (origin) (transition).

That should help set up the rest of the rules. It almost seems like the CFG should be able to generate a string in a backwards sense.

For the rest of the grammar, add these rules to R:

S → X1 Z

Z → {end}

X1 → {start}

C → c

V → v

G) In terms of overall similarity, the rules correlating to transitions are very similar. Since the rules in the previous CFG also follow Chomsky's Normal Form, it's actually very easy to convert the transitions encoded in the grammar of the first CFG to the second. For those transitions in the first CFG such that X# → C/V X#, simply put the second non-terminal X# symbol in front of the rule, then "push" the other X# symbol into the other side of the rule.

i.e.	X1 → V X3	from first CFG
	X3 X1 → V	put the X3 in "front"
	X3 → X1 V	"push" the X1 to the other side

In other words, all the functions of these formats directly correspond to the same transitions, simply in different orders.

In terms of differences, conceptually the first CFG generates strings forwards. This second CFG generates strings backwards. For the inside values of this second CFG, a similar pattern to the inside values of the first CFG would emerge. However, it would be rotated horizontally such that it would be pairs with prefix {start}. Only the second row

corresponding to these aforementioned pairs would have non-zero inside values for X1 to X3. Going back to the truth tables in E), note the characters whose left-hand column for backwards values has a value of 1 for state 1. They would be {epsilon}, c(v)…, and (c)v…. Without bothering to verify them manually, I’m positive that the right-hand forward values for these characters correspond perfectly to the inside values of X1 to X3 in the pairs {start}, {start}c, and {start}v.

The last page is just an inside values table for the new CFG just to manually double-check my assertion above. I guess in terms of stuff I need graded, it ends here.

	… {start}	… c	… v	… c	… {end}
{start} …	S: 0 Z: 0 **X1: 1** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 Z: 0 **X1: 0** **X2: 1** **X3: 0** C: 0 V: 0	S: 0 Z: 0 **X1: 1** **X2: 0** **X3: 1** C: 0 V: 0	S: 0 Z: 0 **X1: 0** **X2: 1** **X3: 0** C: 0 V: 0	S: 1 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0
c …		S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 1 V: 0	S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0
v …			S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 1	S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0	S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0
c …				S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 1 V: 0	S: 0 Z: 0 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0
{end} …					S: 0 Z: 1 **X1: 0** **X2: 0** **X3: 0** C: 0 V: 0