"Unrestricted" grammars

A grammar is a set of variables (some of which are terminals/letters), a start variable, and some production rules

\[(VUT)^+ \rightarrow (VUT)^*\]

\[(V = \text{set of variables}
T = \text{set of letters})\]

E.g.

\[S \rightarrow ABC\]
\[AB \rightarrow xyBA \mid xyB \mid yx\]
\[BC \rightarrow BCax \mid xy \mid Cx\]

So we can have a derivation

\[S \rightarrow ABC \rightarrow xyBC \rightarrow xyxy\]
(Note - you don't have derivation trees anymore)

**Goal:** Prove that every TA language is generated by a grammar, and that any language generated by a grammar is TE (and therefore TA)

1) To show any lang. gen by a grammar is TE - we have to construct a TM that "prints out" all the strings in the grammar.
   - We do this by thinking of all possible derivs. as a tree and doing a BFS (breadth-first-search) on this infinitely tall but finite wide tree.
   - Any result w/ no variables, "print"

Any derivation you will finish at some point, so this "prints" every string gen by grammar.
2) Suppose we have a TA language (i.e., a TM that accepts some strings and rejects or runs infinitely on others). How do I build a grammar that generates the language?

Start in the middle of the process.

1) Idea #1 - changes in instant descr. based on transitions in a TM look like productions in a grammar.

2) Idea #2 - we want to generate all strings (w/ our grammar), but attach some variable tag that says we haven't checked the string yet - run the string through the TM - if it's accepted, remove the variable so that our grammar actually generates the string.
Annoying problem: Our TM probably destroys the input string.

Easy sol’n: Imagine a 2-tape TM, where we just leave the input undisturbed on tape 1 and destroy it all we want on tape 2.

One implementation:

\[ S \rightarrow S V_{le} V^a_{le} \]

(for every letter \( l \) in our input alphabet)

\( V_{le} \) means the tapes on TM have \( \underbrace{l\ldots l} \)

\( (V^k_{le,m} \) means tapes have \( \underbrace{\overbrace{l\ldots l}^{m\times}} \) head is over that spot w/ state \( k \)

This generates all possible input configs - we should start w/ some string to the right of the head which is in state 0.
When we have a transition

\[(q_i, a) \rightarrow (q_j, b, L)\],

we have productions

\[V_{m,p} V_{l,a} \rightarrow V_{m,p} V_{l,b} \quad \text{for every possibility for } l, m, p.\]

For a transition

\[(q_i, a) \rightarrow (q_j, b, R)\],

\[V_{l,a} V_{m,p} \rightarrow V_{l,b} V_{m,p}\]

This lets us get to any possible computation from any possible input.

Now we need to actually recover the input string as a string generated by the grammar if we get to a final state.
Problem we haven't taken care of: the tape has infinite blanks to the side.