The Automata and Computability homework have been assigned to me. This blog post is an outline of the problem with the assumption that you are familiar with automata theory. Automata are models of computation which allow for finite state machines to represent computations, but they are not Turing complete. They can only represent processes which use one input for each output, meaning that they can't model both finite automaton and Turing machine computations simultaneously. The challenge is then how do we model algorithms that create outputs with multiple inputs? This is typically done with a transition system or using help from context free grammars. The homework then asks that we implement a specific state machine. It also provides an answer for this problem using a transition system and context free grammars. I will attempt to provide the solution using the method of implementing a finite state machine and show how it can be converted into both a transition system and context free grammar. The automaton we will look at is as follows: We can see that there are three states denoted as F, T, and U. The possible transitions include Q1 through to Q5 for any input symbol input, where Q2 and Q5 cause the automata to move into state U because they do not accept those states. The states labeled with U in the upper left corner are the only ones that can transition into another state. This is because there is no exit state in our automaton. The free symbols in the upper left corner are the only ones that will be accepted by the automata at any point during execution. Now, given an input symbol, what is the output? We can use this equation to calculate it: This means that if we input 'X' then move to state F with input 'X', we will output 'X'. If we input 'X', move to state F with input 'Y', then output 'X'. This works for any combination of inputs, meaning that there are no strings which lead to no outputs. Note, we can't do this with a regular state machine. We can convert this into a transition system with the following: Now it's easy to see that in order to get any output, you must follow our equation and move in the appropriate direction. For example, given the input string "BZX", we can step through all of the states showing what is outputted when moving into that state: We start in state U because our input doesn't contain 'BZ' (which is free). Then we move into state F which outputs 'BZ', followed by U which outputs 'X', T which outputs 'Z', and finally F again for output of 'X'. We can represent the same transition system using a context free grammar: In this case, we have a single non-terminal 'automata' which represents an output symbol. It has a single rule that depends on the current state and input. This means that our 'automata' can have any output symbol depending on what state it's in and what it's being fed as input. We can convert this into a transition system by replacing the single rule with a set of rules for each possible state/input combination, and then adding edges to the corresponding states in the original automata diagram: The structure is identical to the transition system we represented above; we just replaced type checking with transitions.

448eeb4e9f3229