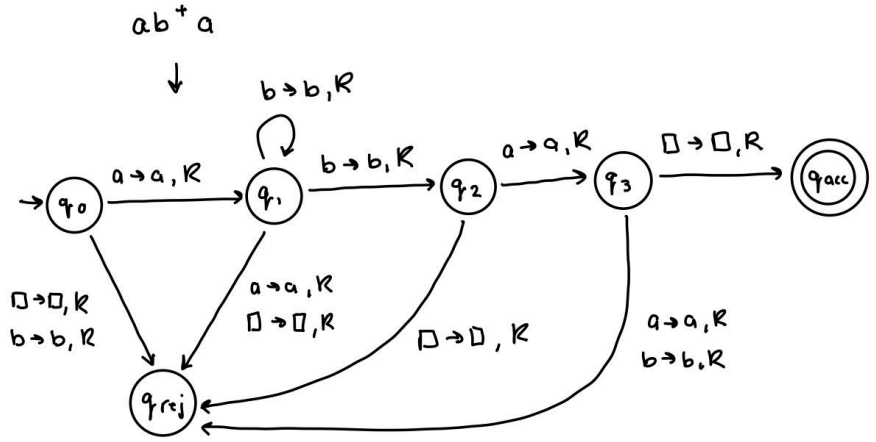
Nancy Norton

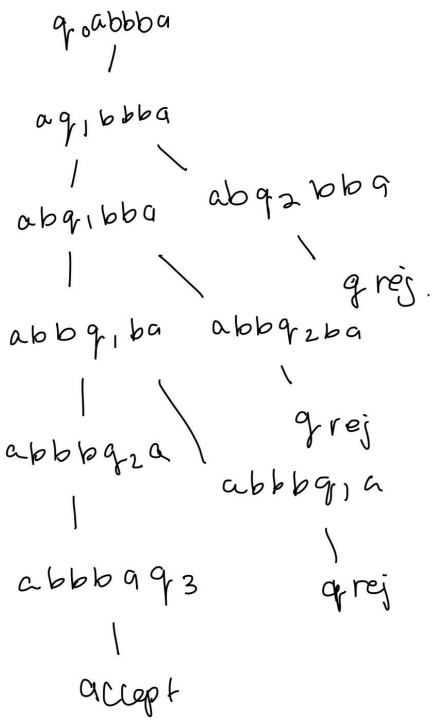
The machine I created is for the language a(b+)a, or the language that accepts any combination that starts with an a, contains one or more b's, and then ends with an a. The nondeterminism in my machine is out of state q1, when there is a b under the tape head, and it transitions either back to q1 or goes to q2. Here is a visual state machine to demonstrate my machine and it's non-determinism:



I wrote my program to take in an input file describing the above machine's transitions. The input file also contains a sample input string to be tested. The program works by first parsing the input file in the parse_input_file() function, then starting from the initial configuration given in that file and running the string given through each transition until it reaches an accepting state or the maximum number of steps is exceeded. Although I don't include the reject state in my transitions in the input files, it can be assumed that any state not explicitly listed leads to a reject state.

In the transition table, for a given state q1 and tape symbol b, the machine may have several possible next states and head positions. The run() function considers all possible transitions from the current configuration and explores each possible transition path by adding all the resulting configurations to the next level of the transition tree. It also keeps count of the number of transitions and steps that have occurred at each level and determines the depth. Each possible transition is treated as a separate branch in the computation tree, and it can branch out into multiple paths at the same time, creating the non-deterministic aspect of the machine.

I determined that my program was correct by using 5 test cases, some of which should be accepted by the machine and some of which should be rejected. I ran these test cases through the state diagram above, drew out possible execution paths and then compared that to the results I got from my code. After a few iterations, I was able to get my written outputs to match the outputs I was getting from the machine, allowing me to determine that my program was working correctly. For example, on the test input string abbba used in inputs3.csv, I drew the following execution path:



When I ran my program on this input, I could then step by step compare each part of the output to determine that my machine was working correctly.

My output traces can be read in the following way: depth, characters on the left of the head character, current state, head character contained within the "| |" symbols, and characters on the right of the head character. At each level, the next character sequentially in the string becomes the head and possible paths from that head are explored, allowing us to clearly follow the trace. For example, the input string "aba" results in the following output trace:

Depth 0: q0|a|ba
Depth 1: aq1|b|a
Depth 2: abq1|a|
Depth 2: abq2|a|
Depth 3: abaq3|_|
Depth 4: aba_qaccept|_|

Here, you can see that at depth 2 there are two possible paths that could be taken to either state q1 or q2.

The degree of nondeterminism I saw varied in each test case, specifically depending on the number of b's that were in a string because this is the character that has two possible transitions in one state and exhibits nondeterminism. For example, in the input string 'aaaa', the degree of nondeterminism is 1 (meaning it is deterministic) because there were no b's present (this string was also rejected because it was not in the language).

Below is the table with a row for each NTM and input string I ran that has the depth, number of configurations explored, and average non-determinism:

Machine: a(b+)a

Input string	result	Depth	# configs explored	Avg nondeterminism = (# configs / depth)
aaaa	rejected	2	2	1
aba	accepted	4	5	1.25
abbba	accepted	6	9	1.5
abbbbb	rejected	7	11	1.57
abbbbbba	accepted	9	15	1.66