NTM	a+ NTM	a+ DTM	{w | w has the same number of 0's and 1's} NTM	{w | w has the same number of 0's and 1's} NTM	a*b*c* NTM	a*b*c* NTM
String used	aaaaa	aaaaa	0011	101	aabc	aabbc
Result	Accepted	Accepted	Accepted	Rejected	Accepted	Ran too long
Depth of tree	7	7	15	6	6	6
# Configs explored	7	7	21	7	7	N/A (transversed 17 transitions before quitting)
Ave. nondeterminism	1.71	1.00	1.05	1.14	2.14	N/A

Comments: I decided to analyze 3 machines (4 if you count the difference between NTM and DTM) and compare outputs of them to each other.

For the a+ machines, I compared the difference between a+ NTM and a+ DTM. In using the same input string I found that the difference between the NTM and DTM machine were the number of configurations explored and the degree of nondeterminism. As expected, the NTM explored more configurations and had a degree of nondeterminism greater than 1.

I used the machine {w | w has the same number of 0's and 1's} NTM to check if my code accepted valid strings and rejected invalid strings. In using the strings ‘01’ and ‘101’, the code ran properly and accepted ‘01’, which has an equal number of 0’s and 1’s, and rejected ‘101’, which has an unequal number of 0’s and 1’s. Both uses resulted in a degree of nondeterminism greater than 1, confirming this machine’s nondeterminism.

Lastly, I used the machine a*b*c* NTM to compare when the depth of the tree is held constant, when a machine couldn’t run. I compared the strings ‘aabc’ and ‘aabbc’ at the depth of 6 and found that while ‘aabc’ ran and explored 7 configurations and accepted in 5 transitions, the addition of one more letter in ‘aabbc’ went further than a tree with a depth of 6 and thus the machine stopped executing the code after 6 steps, traversing 17 transitions before quitting. We can see that with the addition of another character to the machine’s alphabet, the degree of nondeterminism increased compare to the other two machines, a+ and equal 01s.