

# Project 2 Readme Team MJ

12/08/24

1	Team Name: <a href="#">MJ</a>																												
2	Team members names and netids: <a href="#">Max Johnson, mjohns79</a>																												
3	Overall project attempted, with sub-projects: <a href="#">Program 1, Tracing NTM behavior</a>																												
4	Overall success of the project: <a href="#">GOOD! Since I was working by myself, it was easy to get everything done. Outputs all worked and I enjoyed implementing a NTM.</a>																												
5	Approximately total time (in hours) to complete: <a href="#">7 hrs</a>																												
6	Link to github repository: <a href="https://github.com/maxj723/theory-project-two">https://github.com/maxj723/theory-project-two</a>																												
7	<p>List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.</p> <table><tr><th>File/folder name</th><th>File/folder contents and use</th></tr><tr><td colspan="2"><b>Code Files</b></td></tr><tr><td><a href="#">traceTM_MJ.py</a></td><td>It contains all the code to build and run an NTM. Given a CSV, input string, and max depth, it will print the desired output.</td></tr><tr><td colspan="2"><b>Test Files</b></td></tr><tr><td><a href="#">check_tests_MJ.py</a></td><td>Test file to check code output for <a href="#">traceTM_MJ.py</a> run it using <b>python3 check_tests_MJ.py</b></td></tr><tr><td><a href="#">check_MJ folder</a></td><td>Contains all NTM csvs for testing purposes</td></tr><tr><td><a href="#">a_plus_DTM.csv</a></td><td>csv for DTM of aplus language</td></tr><tr><td><a href="#">a_plus.csv</a></td><td>csv for NTM of aplus language</td></tr><tr><td><a href="#">abc_star_DTM.csv</a></td><td>csv for DTM of a*b*c* language</td></tr><tr><td><a href="#">abc_star.csv</a></td><td>csv for NTM of a*b*c* language</td></tr><tr><td><a href="#">equal_01s_DTM.csv</a></td><td>csv for DTM of {w   w has the same number of 0's and 1's}</td></tr><tr><td><a href="#">equal_01s.csv</a></td><td>csv for NTM of {w   w has the same number of 0's and 1's}</td></tr><tr><td colspan="2"><b>Output Files</b></td></tr><tr><td><a href="#">output_MJ folder</a></td><td>Contains all screenshots of testing outputs</td></tr></table>	File/folder name	File/folder contents and use	<b>Code Files</b>		<a href="#">traceTM_MJ.py</a>	It contains all the code to build and run an NTM. Given a CSV, input string, and max depth, it will print the desired output.	<b>Test Files</b>		<a href="#">check_tests_MJ.py</a>	Test file to check code output for <a href="#">traceTM_MJ.py</a> run it using <b>python3 check_tests_MJ.py</b>	<a href="#">check_MJ folder</a>	Contains all NTM csvs for testing purposes	<a href="#">a_plus_DTM.csv</a>	csv for DTM of aplus language	<a href="#">a_plus.csv</a>	csv for NTM of aplus language	<a href="#">abc_star_DTM.csv</a>	csv for DTM of a*b*c* language	<a href="#">abc_star.csv</a>	csv for NTM of a*b*c* language	<a href="#">equal_01s_DTM.csv</a>	csv for DTM of {w   w has the same number of 0's and 1's}	<a href="#">equal_01s.csv</a>	csv for NTM of {w   w has the same number of 0's and 1's}	<b>Output Files</b>		<a href="#">output_MJ folder</a>	Contains all screenshots of testing outputs
File/folder name	File/folder contents and use																												
<b>Code Files</b>																													
<a href="#">traceTM_MJ.py</a>	It contains all the code to build and run an NTM. Given a CSV, input string, and max depth, it will print the desired output.																												
<b>Test Files</b>																													
<a href="#">check_tests_MJ.py</a>	Test file to check code output for <a href="#">traceTM_MJ.py</a> run it using <b>python3 check_tests_MJ.py</b>																												
<a href="#">check_MJ folder</a>	Contains all NTM csvs for testing purposes																												
<a href="#">a_plus_DTM.csv</a>	csv for DTM of aplus language																												
<a href="#">a_plus.csv</a>	csv for NTM of aplus language																												
<a href="#">abc_star_DTM.csv</a>	csv for DTM of a*b*c* language																												
<a href="#">abc_star.csv</a>	csv for NTM of a*b*c* language																												
<a href="#">equal_01s_DTM.csv</a>	csv for DTM of {w   w has the same number of 0's and 1's}																												
<a href="#">equal_01s.csv</a>	csv for NTM of {w   w has the same number of 0's and 1's}																												
<b>Output Files</b>																													
<a href="#">output_MJ folder</a>	Contains all screenshots of testing outputs																												

	output_abcstar_MJ.png	screenshot of output from running different inputs on abcstar NTM
	output_abcstarDTM_MJ.png	screenshot of output from running different inputs on abcstar DTM
	output_aplus_MJ.png	screenshot of output from running different inputs on apus NTM and DTM
	output_equals01s_DTM(1)_MJ.png	screenshot of output from running different inputs on equals01s DTM
	output_equals01s_DTM(2)_MJ.png	more screenshots of output from running different inputs on equals01s DTM
	output_equals01s(1)_MJ.png	screenshot of output from running different inputs on equals01s NTM
	output_equals01s(2)_MJ.png	more screenshots of output from running different inputs on equals01s NTM
	output_test_MJ.png	screenshot of output from check_tests_MJ.py. This just shows that the test works and outputs are correct
8	Programming languages used, and associated libraries: <a href="#">Python – csv, argparse</a>	
9	Key data structures (for each sub-project): <a href="#">NTM class (self made)</a> , <a href="#">Tree (to hold possible configs)</a>	
10	<p>General operation of code (for each subproject):</p> <p>For traceTM_MJ.py, I built an NTM class that contains the attributes and functionality of a non-deterministic turing machine. The class works as follows:</p> <p><b>__init__() function:</b> sets all values of the formal definition tuple given inputs to the attributes of the NTM.</p> <p><b>get_next_config() function:</b> given a config, it determines all possible next configurations using the transition function attribute of the NTM. This is first done by checking the current state and head symbol and identifying if it is possible to transition. Then, it will complete all transitions available and output a new configuration for each. The function returns a list of all the possible configs given. If there are none, it will output a reject config.</p> <p><b>process_input() function:</b> takes an input string and max_depth parameters and runs the turing machine on them. It will print results. The function starts by initializing a tree with the start state as the root config. Then, BFS is used to search the tree for an accepted configuration. When at a certain level, it iterates through each config in that</p>	

	<p>level and checks for accept/reject, then finds all the next possible configs after that state using the <code>get_next_config()</code>. Once it has iterated through all the configs in the layer, it adds the next layer configs to the next layer in the tree until an exit is hit. Then, the output is printed.</p> <p><b>print_output() function:</b> prints the output in the desired way as described by the directions.</p> <p>The main script has a <code>process_csv()</code> function that utilizes the csv library to read the input CSV file and convert it to an NTM object. This NTM object is returned. The main function uses the argparse library to read in cmd line arguments and use the <code>process_csv()</code> function to run the NTM's <code>process_input()</code>. This will result in the desired output.</p> <p>Example usage:  <b>python3 traceTM_MJ.py Test_files/a_plus.csv aaa 7</b>  The first argument after the file name is the NTM CSV file, the second argument is the input string, and the third argument is the max depth.</p>
--	---

1

11	<p>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.</p> <p>The code outputs as dictated in the writeup. When running <code>traceTM_MJ.py</code>, it will first print the name of the machine, then input string, depth reached, total transitions, and whether it was accepted/rejected/maxdepthexceeded. Afterward, it prints out the tree as shown in 3 of the writeup. It will print a level on each line. This means that the last line should contain the accept config.</p> <p>I made 5 different test cases to test my code. First, I used the aplus TM to test because it was provided in the writeup, so I knew that it would be correct for the tests.</p> <p>TEST1 used a short input that would accept but also made it very easy to follow the tree to confirm it was correct.</p> <p>TEST2 used an incorrect character. This would prove that machine would correctly reject</p> <p>TEST3 processed a longer string but with a small max depth. This was to test how the <code>max_depth_exceeded</code> feature would work, and how the output printed as well.</p> <p>TEST4 processed a much longer string just to see how it would fair with longer inputs as well as proper output printing.</p> <p>TEST5 tested a longer input with a bad character. This was to make sure it would still reject.</p> <p>These tests were just to confirm the code worked. I ran the program on other TMs to identify how nondeterminism changes with machines. Please read the results section to get more information on other tests.</p>
----	---

12	<p>How you managed the code development</p> <p>Since I was working alone, it was easier to manage. I developed one function at a time and tried to test it individually. This made it easier to test at the end because I knew each function worked properly as I went. This made it easier to identify what could be causing the problems and how I could fix them.</p> <p>I would have liked to commit my work more often to prevent the possibility of losing saves, though.</p> <p>Using git was convenient for keeping all of the code and tests together.</p>																																																																																				
13	<p>Detailed discussion of results:</p> <p>Table with results for NTM:</p> <table><tr><th>NTM</th><th>String</th><th>Result</th><th>Depth</th><th>#configs</th><th>Avg ND</th></tr><tr><td>a_plus.csv</td><td>aaa</td><td>accept</td><td>4</td><td>10</td><td>2.5</td></tr><tr><td>a_plus_DTM.csv</td><td>aaa</td><td>accept</td><td>4</td><td>4</td><td>1</td></tr><tr><td>a_plus.csv</td><td>b</td><td>reject</td><td>1</td><td>1</td><td>1</td></tr><tr><td>abc_star.csv</td><td>aabbcc</td><td>accept</td><td>7</td><td>47</td><td>6.71</td></tr><tr><td>abc_star.csv</td><td>cc</td><td>accept</td><td>3</td><td>7</td><td>2.33</td></tr><tr><td>abc_star.csv</td><td>aba</td><td>reject</td><td>3</td><td>18</td><td>6</td></tr><tr><td>abc_star_DTM.csv</td><td>cc</td><td>accept</td><td>3</td><td>3</td><td>1</td></tr><tr><td>abc_star_DTM.csv</td><td>aabbcc</td><td>accept</td><td>7</td><td>7</td><td>1</td></tr><tr><td>equal_01s.csv</td><td>0011</td><td>accept</td><td>15</td><td>22</td><td>1.47</td></tr><tr><td>equal_01s.csv</td><td>01010101</td><td>accept</td><td>41</td><td>65</td><td>1.59</td></tr><tr><td>equal_01s.csv</td><td>010</td><td>reject</td><td>6</td><td>9</td><td>1.5</td></tr><tr><td>equal_01s_DTM.csv</td><td>0011</td><td>accept</td><td>15</td><td>15</td><td>1</td></tr><tr><td>equal_01s_DTM.csv</td><td>01010101</td><td>accept</td><td>41</td><td>41</td><td>1</td></tr></table> <p>I chose the three different TMs due to their differing determinism levels. The DTM versions were there to confirm that the code would work and would output a result where depth and transitions are the same.</p> <p>I traced the outputs on each input from above to verify their accuracy, and they all checked out. Additionally, the levels of non-determinism make a lot of sense, verifying that the code works even more.</p> <p>a*b*c* will have higher non-determinism levels because the state diagram has a lot of multiple transitions from one state to another, like when to move on from q0 to q1, etc.</p>	NTM	String	Result	Depth	#configs	Avg ND	a_plus.csv	aaa	accept	4	10	2.5	a_plus_DTM.csv	aaa	accept	4	4	1	a_plus.csv	b	reject	1	1	1	abc_star.csv	aabbcc	accept	7	47	6.71	abc_star.csv	cc	accept	3	7	2.33	abc_star.csv	aba	reject	3	18	6	abc_star_DTM.csv	cc	accept	3	3	1	abc_star_DTM.csv	aabbcc	accept	7	7	1	equal_01s.csv	0011	accept	15	22	1.47	equal_01s.csv	01010101	accept	41	65	1.59	equal_01s.csv	010	reject	6	9	1.5	equal_01s_DTM.csv	0011	accept	15	15	1	equal_01s_DTM.csv	01010101	accept	41	41	1
NTM	String	Result	Depth	#configs	Avg ND																																																																																
a_plus.csv	aaa	accept	4	10	2.5																																																																																
a_plus_DTM.csv	aaa	accept	4	4	1																																																																																
a_plus.csv	b	reject	1	1	1																																																																																
abc_star.csv	aabbcc	accept	7	47	6.71																																																																																
abc_star.csv	cc	accept	3	7	2.33																																																																																
abc_star.csv	aba	reject	3	18	6																																																																																
abc_star_DTM.csv	cc	accept	3	3	1																																																																																
abc_star_DTM.csv	aabbcc	accept	7	7	1																																																																																
equal_01s.csv	0011	accept	15	22	1.47																																																																																
equal_01s.csv	01010101	accept	41	65	1.59																																																																																
equal_01s.csv	010	reject	6	9	1.5																																																																																
equal_01s_DTM.csv	0011	accept	15	15	1																																																																																
equal_01s_DTM.csv	01010101	accept	41	41	1																																																																																

	<p>This is shown in the output and average nd level. On the other hand, equal_01s seems to have less non-determinism in this case due to the highly systematic process of finding 1s and 0s.</p> <p>The average nondeterminism stat is interesting because regardless of how long the input or how deep the search went, the average nondeterminism for NTMs stayed relatively the same. This shows that the results depend much more heavily on the language and the machine rather than the input or input length.</p>
14	How team was organized: Team only consisted of me, so I did all the work
15	What you might do differently if you did the project again: I would probably work with a partner so that I would have more time to explore the deeper options and have fun experimenting with other ideas.
16	Any additional material: