Project 2 Readme Team hflick

Version 1 12/5/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

<table>
<tr><td>1</td><td>Team Name: hflick</td></tr>
<tr><td>2</td><td>Team members names and netids: Hunter Flick (hflick)</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects: Tracing NTM Behavior</td></tr>
<tr><td>4</td><td>Overall success of the project: Successfully traced execution of NTM machines a+ and a*b*c* with a variety of input strings and maximum depths</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete: 11 hours</td></tr>
<tr><td>6</td><td>Link to github repository: https://github.com/huntflick/TOC-NTM-Tracing-hflick</td></tr>
<tr><td>7</td><td>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.
<table>
<tr><td>File/folder Name</td><td>File Contents and Use</td></tr>
<tr><td colspan="2">Code Files</td></tr>
<tr><td>traceTM_hflick.py</td><td>Contains the NTM tracer, consisting of a main function to parse command line inputs and the csv file, a function to print results, and a recursive tracer that evaluates paths</td></tr>
<tr><td colspan="2">Test Files</td></tr>
<tr><td>dataCheck_a+_hflick.csv</td><td>CSV file that describes an NTM that accepts one or more a's (a+)</td></tr>
<tr><td>dataCheck_abcstars_hflick.csv</td><td>CSV file that describes an NTM that accepts any number of a's, b's, and/or c's in that order (a*b*c*)</td></tr>
<tr><td colspan="2">Output Files</td></tr>
<tr><td>output_a+_hflick.txt</td><td>Results from running program on a+ NTM and six input strings</td></tr>
<tr><td>output_abcstars_hflick.txt</td><td>Results from running program on a*b*c* NTM and</td></tr>
</table>
</td></tr>
</table>

	<table><tr><td></td><td>eight input strings</td></tr><tr><td>output_tabular_hflick.pdf</td><td>Above results in tabular form</td></tr></table>
8	Programming languages used, and associated libraries: Languages: Python Libraries: csv, sys
9	Key data structures (for each sub-project): The key data structure I used was a 2D list. I used lists for the rules, levels of the execution tree, and various paths. The creation of the rule list was done by reading in the csv line by line, appending new rules. Levels of the tree were created by adding all possible transitions to a list. The paths were created by appending configurations generated by corresponding rules to already existing paths. Using 2D lists was incredibly useful as it made storing paths and calculating depth easy through indexing and list slicing.. It also provided a clean way to group configurations while also retaining their status as independent states.
10	General operation of code (for each subproject): The program begins by parsing the command line and ensuring that the user provided a valid input. Then the csv file is parsed, which gives the list of rules, machine name, and start, reject, and accept states. The program then calls a sub function inside of a while loop, which allows for multiple input strings to be processed in the same run. The called function initializes the initial configuration and calls the recursive path tracer. The recursive function cycles through all rules for the NTM and builds a list of configurations to visit. The function returns if the maximum transition limit is reached, the NTM transitions to reject, or if the NTM transitions to accept. If the maximum transition limit is reached or the NTM accepts, no more paths are explored. If the NTM rejects, the function determines whether the new reject path is longer than the longest current reject path. Then, the program backtracks and continues exploring new paths. Once the path tracing is complete, the function returns and the results for that input string are printed. Throughout the program, I made use of Python's try/except blocks to handle indexing errors for lists.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code: I used CSV files describing machines a+ and a*b*c* to test my code. I used a+ for much of my initial testing as I could easily verify the transitions using the diagram provided in the project document. Once my program worked with a+ and a wide variety of input strings for the machine, I moved onto a*b*c* and continued testing. I modified the files by making transitions to accept move the head left and cleaned up the formatting by removing extra spaces and unused states from the source files.
12	How you managed the code development: I began by writing the main() function to parse the command line and csv file to ensure that the program would only run on valid inputs for later testing. I then wrote a function that generated the entire tree. I used this intermediate function to ensure I had a good grasp on how to trace an NTM. Once I was certain the output was correct, I adapted this

	function and used it to develop the recursive function. From there, I explored where to put the base cases and return statements by running the code on an input that I had already evaluated by hand. Throughout the early development, I used the csv for the a+ NTM and the string 'aaaa'.
13	Detailed discussion of results: The general trend between the two machines is that as the length of the input strings increases, the nondeterminism and number of configurations explored increases. This is because as the string length increases, the machine naturally has more options of paths to explore. The output from the machine for $a^*b^*c^*$ illustrates this trend for the number of configurations explored. It also shows a rough trend of nondeterminism increasing as well. An exception to this rule is when the string has some character that causes the machine to reject early in the processing. For example, the string 'abaa' has the same length as the string 'aaaa', but has the same results as the string 'a' when put into the machine for a^+. This is because the machine will follow the same paths of execution for both strings.
14	How team was organized: I completed the project on my own and was in charge of all development and testing.
15	What you might do differently if you did the project again: If I did the project again, I would write extensive pseudocode to ensure I completely understood the recursive calls from the onset of the project. This would have cut down on the time I spent to test and understand what values my functions were returning.
16	Any additional material: N/A

a^+ NTM, Maximum Transitions 15					
Input String	Result	Tree Depth	Configurations Explored	Average Nondeterminism	Comments
aaaa	Accepted	5	11	$1.\overline{6}$	String in the language
a	Accepted	2	5	$1.\overline{3}$	String in the language
_	Rejected	1	2	1.0	String not in the language
aaab	Rejected	4	11	$1.\overline{428571}$	String not in the language
abaa	Rejected	2	5	$1.\overline{3}$	String not in the language
aaaaaa	Accepted	7	15	1.75	String in the language

$a^*b^*c^*$ NTM, Maximum Transitions 25					
Input String	Result	Tree Depth	Configurations Explored	Average Nondeterminism	Comments
aaaa	Accepted	6	19	3.0	String in the language
a	Accepted	3	7	2.0	String in the language
_	Accepted	2	3	1.0	String in the language
aaab	Accepted	5	19	$2.\overline{571428}$	String in the language
abaa	Rejected	3	19	$1.\overline{63}$	String not in the language
aaabbc	Accepted	7	24	$2.\overline{5}$	String in the language
acc	Accepted	4	11	2.0	String in the language
aaaaabbccb	Timeout	Unknown	25	$3.\overline{714285}$	String would eventually be rejected