Project 2 Readme Team DishyWizard

12/11/25

1	Team Name: DishyWizard
2	Team members names and netids: Jack Frauenhofer (jfrauenh)
3	Overall project attempted, with sub-projects: ntm_tracer
4	Overall success of the project: Successful
5	Approximately total time (in hours) to complete: 12 hours
6	Link to github repository: https://github.com/jackFrauenhofer/Project2-TOC
7	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.

File/folder Name	File Contents and Use
Code Files	
ntm_tracer.py	This file contains the function to run NTM_tracer, as well as print out the tree.
Test Files	
(All files below are saved in /input folder) Aplus.csv Composite.csv Equal_01s.csv At_least_3-0s_or_3-1s.csv	These csv files were used to test the ntm tracer. Aplus, composite, and equal_01s were given by prior students, where as I created at_least_3-0s_or_3-1s to further test how the machine works.
Output Files	
(All files below are saved in /output folder) Aplus_output Composite_output Equal_01s_output At_least_3-0s_or_3-1s_output	These files print the traced tree that the NTM explores when running through specific inputs.

<table>
<tr><td></td><td>
<table>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>N/A</td><td>No plots were needed</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:

Python
Csv
Sys
argparse</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):

Config: Config was a list structure that stored different states of the NTM machine. Its syntax is [left_stack, current_state, right_stack]. By using this, it was easy to save specific transition states that the NTM machine could be in and also made it easy to print it out.

Tree: Tree was a list of lists of lists, as specified in the project instructions. In fact, it was a list of configs, where each config was then grouped into specific lists that separated them by depth.

Transitions: This is a list of transitions that a machine can take when it is in a specific state. When the NTM machine runs, it loops through these transitions in order to explore possible branches that could result in an accept state. If accepted, the machine ends, and if it rejects, a new transition is explored.</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject):

Ntm_tracer:

My ntm_tracer runs in the fashion outlined in the project instructions. The tracer uses the provided ‘get_transitions’ functions in order to, at each configuration, find every possible transition the machine can take given the current state and the head of the tape. By doing this, the machine can replicate nondeterminism by exploring several different branches / transition paths the machine can take simultaneously. It does this by doing a BFS-esque search through this generated tree.

When evaluating each configuration, it checks first whether the configuration is in an accept or reject state. If in an accept state, the tracer actually returns, because an accepted path was found. If rejected, the tracer just kills that branch and continues to explore the other possible branches, if any. If there are none, the machine rejects. In order to eliminate halting issues, the machine can set different max_depths in order to fix how far the tracer should search different paths.

The most difficult aspect of this program to build was, for each transition a configuration can take, there are different ways to manipulate the left and right stack depending on the direction of the transition. The program does this by first rewriting the head, and</td></tr>
</table>

	then, depending on the direction, adds it to the right/left stack, and then replacing the head. There were many cases where the program would fail in testing because there would be no inputs left on either stack. To handle this, I had to add many if-else statements to handle various cases where the stacks could have characters or be blank. **Print_tree:** In order to print the tree, I created a function that, given the tree data structure. Prints each configuration in an easy to read way. By printing at each depth level and on separate lines, this function allows a user to easily analyze the effectiveness and nondeterminism of the NTM.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I used aplus.csv to first check the correctness of my code. This was my first test because of the simplistic nature of the NTM that only accepts one character and really only uses 1-2 transition paths. After I verified my tracer works on aplus.csv, I then tested on equal_01s.csv. This was a great test to test various different inputs that could trigger different transitions. At first, there were several bugs in my tracer, especially when the stack was blank, which prompted me to add edge cases. Lastly, I built the at_least_3-0s_or_3-1s.csv, which is based on a NFA that we built early this semester. I also used the composite.csv file for the final tests to verify my tracer was effective. Once I ran these, I felt confident that my tracer worked on any correct NTM transition path file.
12	How you managed the code development I managed the code development by a series of trial and error. I was a solo team, so every detail in the project I built. I largely attacked this project over the course of Monday and Tuesday of this week. I first read the instructions for project 2, and then outlined how I would build such a program. With this as a guide and Lax's github cloned, I was able to build the program relatively easily. A large issue I faced handling the different transition directions and how they manipulate the different stacks. To do this, I utilized many print() statements in order to consistently verify what the left stack, right stack, and head looked like. By doing this, I was able to easily track where inconsistencies lied. This was especially useful when I realized the n1n.csv file was malfunctioning.
13	Detailed discussion of results: Overall, I thought this project was a great learning experience and a success. The ntm_tracer is able to track every plausible transition and find an accepted path using BFS. Invalid inputs give an error, and valid inputs are either accepted, rejected, or hit the max_depth. I calculated the degree of nondeterminism by dividing the total number of

	transitions by the number of non-leaf configurations in the tree.
14	How team was organized: Solo team
15	What you might do differently if you did the project again: If I were to do the project differently again, I would first try to improve the readability / syntax of handling the different directions of possible transitions. I think how I built this was overly complex with many different edge cases, and I would explore easier ways to solve this. I would also try to build a function that, after printing out the entire tree, would also backstep from the accepted state to show the transition path that led to this accepted state. I think this would be a helpful distinction to add to highlight the accepted path amidst the complex configurations the entire nondeterministic tree stores.
16	Any additional material: None