Project Readme

12/8/2024

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"

1	Team Name: warmswor
2	Team members names and netids: Will Armsworthy, warmswor
3	Overall project attempted, with sub-projects: Project 1 - Tracing NTM Behavior
4	Overall success of the project: success
5	Approximately total time (in hours) to complete: 8 hours
6	Link to github repository: https://github.com/willarms/TOC-Project2
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	
traceTM_warmswor.ipynb	Contains all code/testing code for each test file and test case.
Test Files	
aplus.csv	Contains machine csv for aplus test case
abc_star.csv	Contains machine csv for abc_star test case
Output Files	
output_aplus.txt	Contains all output from code when running aplus.csv test file. This includes info about machine (name, states, etc.) and accept/reject

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>as well as path and determinism metric.</td></tr>
<tr><td>output_abc_star.txt</td><td>Contains all output from code when running abc_star.csv test file. This includes info about machine (name, states, etc.) and accept/reject as well as path and determinism metric.</td></tr>
<tr><td colspan="2">Plots (as needed)</td></tr>
<tr><td>N/A</td><td>N/A</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>Programming languages used, and associated libraries:
Python - associated libraries: csv and google.drive (to mount drive)</td></tr>
<tr><td>9</td><td>Key data structures (for each sub-project):
transitions → dictionary used to keep track of all states in csv
state_sequence → dictionary used to keep track of all the states in a given path
transition_list → list used to keep track of the counts of transitions for each level (used to compute determinism metric)</td></tr>
<tr><td>10</td><td>General operation of code (for each subproject)
Project 1:
Can see discussion of results for more detail, but essentially my code is split up into 2 functions and a main section. The first function is simply a helper for the second, it helps to print the sequence of states and transitions if a string is accepted. The second function does the simulating, as it takes in the csv file, parses it to build a transitions dictionary, and then iterates through that dictionary using the input string to identify an accept state, a reject state, or reach a max_depth/max_transitions number where it automatically rejects. This function is then called in the main section along with a couple different test cases (there is a flag to toggle between which test case is used).</td></tr>
<tr><td>11</td><td>What test cases you used/added, why you used them, what did they tell you about the correctness of your code.

Again, see detailed discussion of results for more info on this, but I used the aplus.csv and abc_star.csv from the class files. These were used because they helped me during the development process (since they were simple enough where I myself could check if an input string should be accepted or not), and because they display differing levels of nondeterminism. They were able to fix some of the errors (all of the errors actually) that I was facing with regard to parsing input, constructing my data structures, and printing the steps taken to an accept state.</td></tr>
<tr><td>12</td><td>How you managed the code development
I started by building a function to parse the input file for the aplus.csv machine. Once I</td></tr>
</table>

	had this, I came up with a sample input string and tried to come up with a way to parse it and get what my machine would do. This didn't work and I had to come up with a way to store the transitions in a way that would be easier to go through and check what the machine would do upon different inputs. I did this by consulting the program document again and using the tree method with lists of lists for each possible route. This helped greatly, and I was able to figure out the correct process for the aplus example. I then developed my testing/output setup, iterated on these when I tried my code for abc_star, and finished up the small errors that I experienced during this second test case.
13	Detailed discussion of results: **(can also look at code_analysis.txt in github)** My code works by first parsing the input file. I assume that the input files mirror the behavior described in the document and shown in the aplus.csv sample, which I use as my first input case. I use the csv library from python to parse the file after loading it in from my drive folder. Next, I gather the metadata from the top of the file. This includes the machine name, states, accept state, reject state, and start state. Then, I construct a dictionary called "transitions" which holds each state and potential next states as key,values, like it is described in the project instructions. This dictionary is then parsed through in the next part of my NTM_Tracer function, which looks at the state, character, next state, character to write, and direction to move the tape for each possibility. It keeps track of the sequence of states in a dictionary (to return if accepted), and calculates the determinism metric, explained in the project instructions, that I will discuss at the end of the document. Finally, the NTM function goes through each possibility, evaluating whether accept, reject, or too many states will be reached. In each case, the function returns a boolean with the result of the input string and information about whether it was accepted (and the path) or how many states it was rejected in. Looking at the test cases I used to determine the functionality of my program, I used the aplus.csv and abc_star.csv machines (both found in the class folder). Using my knowledge of NTM/DTMs and the work we have done in class so far, I was able to make sure that these would make sense to use and be valid test cases.

	Next, I came up with 5 test input strings for each, shown as input strings #s 1-5 in my code. These were designed so that the first should reject, second and third should accept, fourth should reject, and fifth should accept (shown in output txts for both). To verify that my code worked, I made sure that all these line up, along with a couple other input strings that I then removed from my file for easier reading. Additionally, my program computes a determinism metric for each input string. To calculate this metric, I had to carefully analyze how to trace through each input string. To more fully show how an input string is handled, on a string that was accepted, I print out the path that the machine took to get there, in line with our homeworks from class. This path (or errors in the path), helped me debug my program and make sure that it was acting in the way that I wanted it to. Additionally, it helped me verify that I was calculating the determinism metric correctly, by averaging the number of potential transitions from each step, and then iterating on this as I go through. Looking at the determinism metric (I'll call it d metric for simplicity) ratings for each of my test cases, we can see that the first machine that I looked at (aplus), shows a moderate level of nondeterminism. Particularly if you look at the strings that accept (2, 3, and 5), they all have d metrics of 1.8 or higher, showing a relatively high level of nondeterminism. This makes sense because the NTM works by having multiple transitions on q1 given the input of an a, and relies on nondeterminism to know whether to stay on q1 or move to q2. Conversely, if you look at the outputs for the second machine that I looked at (abc_star), they show that this machine displays slightly less determinism than the aplus machine. Particularly, the input string abc, that cleaning flows through the states of the machine, actually has a determinism metric of 1.0, because at no point are there multiple possibilities for an input. While this 1.0 rating doesn't exist for the other input strings that are a little more complex, none of them reach a d metric of

	higher than about 1.3 (if they are accepted). This goes to show that this machine shows considerably more determinism than the previous machine analyzed, and also goes to validating the structure of my code and the logic of my computations.
14	How team was organized: individual project
15	What you might do differently if you did the project again: Use additional testing machines to check for more extreme cases on nondeterminism
16	Any additional material: None