Project 2 Readme Team cmadon4

Version 1 9/11/24

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: cmacdon4</td></tr>
<tr><td>2</td><td>Team members names and netids

Connor MacDonald
cmacdon4</td></tr>
<tr><td>3</td><td>Overall project attempted, with sub-projects:

Program 1</td></tr>
<tr><td>4</td><td>Overall success of the project:

perfect!</td></tr>
<tr><td>5</td><td>Approximately total time (in hours) to complete:

I spent about 12 hours.</td></tr>
<tr><td>6</td><td>Link to github repository:

https://github.com/cmacdon4/Theory_of_Computing_Project_02_cmacdon4</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_cmacdon4.py</td><td>Hamiltonian Path checker

/traceTM_cmacdon4.py $File $String $termination_flag</td></tr>
</table>
</td></tr>
</table>

Test Files	
check_a_plus_DTM_cmacdon4.csv	DTM for the a+ regex Use for traceTM_cmacdon4.py
check_a_plus_cmacdon4.csv	NTM for the a+ regex Use for traceTM_cmacdon4.py
check_abc_star_DTM_cmacdon4.csv	DTM for the a*b*c* regex Use for traceTM_cmacdon4.py
check_abc_star_cmacdon4.csv	NTM for the a*b*c* regex Use for traceTM_cmacdon4.py
check_equal_01s_DTM_cmacdon4.csv	DTM for the 0^n1^n regex Use for traceTM_cmacdon4.py
check_equal_01s_cmacdon4.csv	NTM for the 0^n1^n regex Use for traceTM_cmacdon4.py
Output Files	
output_a_plus_NTM_accept_cmacdon4.png	Terminal output for associated TM. See image for associated

	inputs
output_abc_star_DTM_accept_cmacdon4.png	Terminal output for associated TM. See image for associated inputs
output_abc_star_NTM_accept_cmacdon4.png	Terminal output for associated TM. See image for associated inputs
output_abc_star_NTM_reject_cmacdon4.png	Terminal output for associated TM. See image for associated inputs
Plots	
plots_results_table_cmadon4.png	Results of several runs

8 Programming languages used, and associated libraries:

Python; OS, sys

9 Key data structures (for each sub-project):

To construct the tree I used a list of lists of tuples. Each list represents a level on the tree. Each tuple is a configuration that is formatted as follows: (input_seen, curr_state, input_next, prev_state). This allows for backtracking by matching the child's previous state with the level above's current state. Here is an example of what a tree could look like

```
[('', 'q0', 'abc_', None)]
[('a', 'q0', 'bc_', 'q0'), ('a', 'q1', 'bc_', 'q0'), ('a', 'q2', 'bc_', 'q0'), ('a', 'q3', 'bc_', 'q0')]
[('ab', 'q1', 'c_', 'q0'), ('ab', 'q2', 'c_', 'q0'), ('ab', 'q3', 'c_', 'q0'), ('ab', 'q1', 'c_', 'q1'), ('ab', 'q2', 'c_', 'q1'), ('ab',
 'q3', 'c_', 'q1')]
[('abc', 'q2', '_', 'q2'), ('abc', 'q3', '_', 'q2'), ('abc', 'q2', '_', 'q2'), ('abc', 'q3', '_', 'q2')]
[('abc_', 'qacc', '', 'q3')]
```

Figure 1 – tree for check_abc_star_cmacdon4.csv on input abc

Because the tree is non-deterministic, its levels usually increase in the number of configurations until accepted.

10	General operation of code (for each subproject) Given a Turing Machine (either deterministic or nondeterministic), an input string, and a termination flag, my code creates a tree of all the paths that the machine could take. This required parsing a CSV file, which I made into an object of the Turning Machine class, which I also defined. To create the tree, I used a breadth-first search, branching off the valid transitions. After a tree is produced, it can be backtracked to define the path to reach the accept state.
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I used the following tests from the course website: check_equal_01s_cmacdon4.csv check_equal_01s_DTM_cmacdon4.csv check_abc_star_cmacdon4.csv check_abc_star_DTM_cmacdon4.csv check_a_plus_cmacdon4.csv check_a_plus_DTM_cmacdon4.csv I used these programs because they were convenient and reliable. To make sure the programs were reliable, I would draw them by hand. Because I drew them out, I was then able to walk through my tree and the correct path with the Turing machine to ensure correctness. This ended up being a powerful way to test my code.
12	How you managed the code development The development of this code involved an iterative approach to ensure clarity and functionality. First, I analyzed the requirements, focusing on creating a program capable of parsing a Turing Machine description from a file and simulating its execution, whether deterministic or nondeterministic. I started by implementing the CTM class to encapsulate the machine's properties, such as states, transitions, and configurations, enabling clean data handling. Next, I developed utility functions like parse_csv for reading the Turing Machine definition and usage for user guidance and error handling. The simulation logic in TM_walk required careful design to handle both deterministic and nondeterministic cases, using a breadth-first search to explore possible paths and limiting execution based on the user-defined termination flag. I included a backtrace function to reconstruct the path leading to acceptance or rejection and calculated metrics like the degree of nondeterminism for insight into the simulation. Finally, I implemented the output function to summarize results clearly and informatively. Throughout development, I focused on modularity, ensuring each function handled a specific task, and used test cases to validate the program's behavior under various scenarios.

13	Detailed discussion of results:

The program operated nominally. For all the test cases given various input strings, the following table was produced:

Table 1 – Results of several runs

TM	Nondeterministic?	Input String Result	Result	Depth of Tree	#Transitions Simulated	Average Nondeterminism
check_abc_star_cmacdon4.csv	TRUE	abc_	accept	4	15	3.2
check_equal_01s_cmacdon4.csv	TRUE	101010_	accept	25	38	1.46
check_a_plus_cmacdon4.csv	TRUE	aaaaa_	accept	6	11	1.71
check_a_plus_cmacdon4.csv	TRUE	_	reject	0		
check_abc_star_DTM_cmacdon4.csv	FALSE	abc_	accept	4	4	1
check_abc_star_DTM_cmacdon4.csv	FALSE	abcc_	reject	3	4	1
check_equal_01s_DTM_cmacdon4.csv	FALSE	010101_	accept	25	25	1
check_a_plus_DTM_cmacdon4.csv	FALSE	aaaaa_	accept	6	6	1

From what is gathered in the table and the screenshots given in the repository, the program is adequately able to trace both DTMs and NTMs without a problem. As discussed, these tests and the results were verified by hand tracing.

From the table and other results, I noticed that the average nondeterminism for NTMs was a higher number as the input string got longer. This makes sense because there are more possible options to take. There is an average determinism of 1 for DTMs because there is only one possible transition.

The program likely ran at O(n^2) because of the nested for loops that I used. To traverse the data structures used, however, this could not be avoided.

14	How team was organized I worked alone
15	What you might do differently if you did the project again I might reduce the memory overhead by doing everything on the fly instead of storing the data in a class.
16	Any additional material: I do not!