Project 2 Readme Team gargueta

Version 1 9/11/24

1	Team Name: gargueta
2	Team members names and netids: Genesis Argueta - gargueta
3	Overall project attempted, with sub-projects: Tracing NTM Behavior
4	Overall success of the project: Completed and Sucessful
5	Approximately total time (in hours) to complete: 8
6	Link to github repository: https://github.com/g3n3s1s-a/traceTM_gargueta
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	
code_gargueta.py	This is the main file where we parse a csv file to extract the formal description of the non-deterministic turing machine and a function that simulates a non-deterministic turing machine.
Test Files	
A_plus_accept_gargueta.csv Aa_plus_bb_accept_1_gargueta.csv Aa_plus_bb_accept_2_gargueta.csv Aa_plus_bb_reject_gargueta.csv One_plus_zero_accept_1_gargueta.csv One_plus_zero_accept_2_gargueta.csv one_plus_zero_reject_gargueta.csv	This is where all the csv files are located. There are 3 different machines tested with different strings. The csv files are organized in the following matter: - Name of machine - String used

	<ul><li>List of state names for Q</li><li>List of characters from Σ</li><li>List of characters from Γ</li><li>The start state</li><li>Accept state</li><li>Reject state</li></ul>
Output Files	
A_plus_accept_gargueta.pdf Aa_plus_bb_accept_1_gargueta.pdf Aa_plus_bb_accept_2_gargueta.pdf Aa_plus_bb_reject_gargueta.pdf One_plus_zero_accept_1_gargueta.pdf One_plus_zero_accept_2_gargueta.pdf one_plus_zero_reject_gargueta.pdf	Output files that show the name of the machine used, string processed, configurations at each depth, level of non-deterministic, and transition log.
Plots (as needed)	
table_gargueta.pdf	Table with the following information for each machine:<ul><li>Name</li><li>input string</li><li>Result</li><li>Depth</li><li># configurations explored</li><li>Average non-determinism</li></ul>

8	Programming languages used, and associated libraries: Python, csv, collections
9	Key data structures (for each sub-project): tree and dictionary
10	General operation of code (for each subproject) : This code simulates a nondeterministic Turing Machine (NTM) defined by a CSV file. The parse_csv function reads the machine's configuration, including states, transitions, and the input string. The simulate_ntm function explores all possible configurations of the machine, using breadth-first search to process transitions. It tracks the total, accepted, and rejected configurations, as well as the level of nondeterminism, until it either accepts, rejects, or reaches a depth limit. The program outputs detailed logs of configurations and transitions during execution.

11 What test cases you used/added, why you used them, what did they tell you about the correctness of your code.

1. a_plus_accept_gargueta.csv
    a. Input: aaa
    b. Purpose: To test a simple machine that accepts strings matching the pattern a*.
    c. Result: The machine correctly accepted valid strings and identified all configurations, demonstrating its ability to handle repetition of a single character.
2. aa_plus_bb_accept_1_gargueta.csv
    a. Input: aaabb
    b. Purpose: To verify the machine correctly handles concatenated patterns (a+b)∗(aa+bb)(a+b)∗.
    c. Result: The machine successfully accepted valid strings, showing its correctness for nested patterns.
3. Aa_plus_bb_accept_2_gargueta.csv
    a. Input: aabb
    b. Purpose: To check if the machine can process variations of the pattern (a+b)∗(aa+bb)(a+b)∗ without trailing characters.
    c. Result: It accepted the string as expected, validating flexibility in the middle substring pattern.
4. Aa_plus_bb_reject_gargueta.csv
    a. Input: abab
    b. Purpose: To confirm the machine rejects invalid patterns that don't conform to (a+b)∗(aa+bb)(a+b)∗.
    c. Result: The machine correctly rejected the input, proving it can distinguish invalid strings.
5. One_plus_zero_accept_1_gargueta.csv
    a. Input: 10010
    b. Purpose: To test a machine with a more complex binary pattern 1(0+1)∗ 0.
    c. Result: The machine accurately accepted the input, confirming it can process binary strings with a valid prefix and suffix.
6. One_plus_zero_accept_2_gargueta.csv
    a. Input: 10
    b. Purpose: To verify if shorter binary strings are correctly processed.
    c. Result: It accepted the input, proving it handles minimal valid cases.
7. One_plus_zero_reject_gargueta.csv
    a. Input: 10011
    b. Purpose: To ensure invalid binary strings are rejected by 1 (0 + 1)∗0
    c. Result: The rejection confirmed the machine's ability to identify mismatches.

12	How you managed the code development I managed the code development using git and github. I consistently followed a structured workflow to ensure smooth progress. I also added debugging statements in my code to ensure validity of the functions.
13	Detailed discussion of results: The simulator successfully traced all possible paths for a nondeterministic Turing Machine (NTM) using breadth-first exploration, ensuring correctness and preventing infinite loops. The results show: Correctness: Accepting and rejecting states were accurately identified across all test cases, including valid strings (e.g., aaa for A_plus) and invalid strings (e.g., abab for Aa_plus_bb). Breadth-First Traversal: Enabled exhaustive path exploration, capturing all configurations and ensuring acceptance decisions weren't missed. Metrics: Depth, configurations, and average nondeterminism reflected the complexity of the machines and verified the implementation's alignment with theoretical behavior. Challenges: High branching factors increased resource usage, and setting an appropriate depth limit was crucial to balance completeness and performance. The max depth was 20. The results validate the simulator's correctness, reliability, and adherence to the project goals.
14	How team was organized: I worked individually.
15	What you might do differently if you did the project again If I did the project again, I'd use a class to better organize the machine's attributes and methods, improving readability and modularity. I'd also separate parsing, simulation, and logging into distinct components and add type annotations for clarity and error handling for robustness.
16	Any additional material: