Readme_DaniaBenecke

Version 1 8/22/24

A copy of this file should be included in the github repository with your project. Change the teamname above to your own

1. **Team name:** DaniaBenecke
2. **Names of all team members:** Dania Benecke
3. **Link to github repository:**git@github.com:dbenecke/Theory-of-Computing-Project-2.git
4. **Which project options were attempted:** Program 1: Tracing NTM Behavior
5. **Approximately total time spent on project:** 13 hours
6. **The language you used, and a list of libraries you invoked.**
    a. Language: Python
    b. Libraries: Csv and from collections deque
7. **How would a TA run your program (did you provide a script to run a test case?)**
    a. I provided 5 test scripts. The Test script check4_DaniaBenecke.csv is already put down as the test file in the code but it can be changed. The input string is selected inside the code, also already provided to the TA as abc, but can also be changed.
8. **A brief description of the key data structures you used, and how the program functioned.**
    a. Key Data Structures
        i. List of Dictionaries: Used to store transitions, where each dictionary represents a state transition rule (current state, input symbol, next state, tape symbol replacement, and head movement).
        ii. Deque: Utilized to manage paths of configurations in a breadth-first way, to explore all the nondeterministic branches.
        iii. Strings: Represented the left and right parts of the tape along with the head's current state for each configuration.
    b. How does it function
        i. Reads Turing machine specifications and transitions from a CSV file.
        ii. Simulates the operation of a nondeterministic Turing machine by exploring all possible paths up to a specified depth (maxD).
        iii. The heart of the program is the run method, which simulates the NTM's operation. This involves:
            1. Queue-Based Breadth-First Search: The program uses a deque to manage configurations in a breadth-first way, ensuring all possible paths are explored
            2. Each configuration includes:
                a. The left part of the tape.
                b. The current state.
                c. The right part of the tape (including the head).

3. Depth Control: To prevent infinite exploration since the TM is nondeterministic, the program imposes a maximum depth (maxD). Any path exceeding this limit is stopped.
4. Transition Evaluation: For the current configuration, the program evaluates all valid transitions.
    a. A transition is valid if:
        i. The current state matches the transition's "nowstate."
        ii. The symbol under the tape head matches the transition's input symbol.
        iii. Each valid transition generates a new configuration, which is added to the queue for further exploration.
5. Keeps track of paths and stops when an accepted state is reached or all possibilities are exhausted and thus rejected has been reached.

9. **A discussion as to what test cases you added and why you decided to add them (what did they tell you about the correctness of your code). Where did the data come from? (course website, handcrafted, a data generator, other)**
    a. Test cases added:
        i. check1_DaniaBenecke.csv
        ii. check2_DaniaBenecke.csv
        iii. check3_DaniaBenecke.csv
        iv. check4_DaniaBenecke.csv
        v. check5_DaniaBenecke.csv
    b. Why I added them
        i. What it tells me about correctness of code: each test case tries a different turing machine for the program. This ensures that the program works for any Turing machine and not just for 1. It also allows me to try multiple input strings ensuring the consistency of the output.
    c. Where the data came from: Professor Kogge's test files
10. **An analysis of the results, such as if timings were called for, which plots showed what? What was the approximate complexity of your program?**
    a. Plots were not used for this project. The complexity is determined by the degree of nondeterminism each machine and input string have.
11. **A description of how you managed the code development and testing.**
    a. Management of code development
        i. I first started by rereading how to write configurations
        ii. I then watched a couple of BFS youtube videos to recall the algorithm
        iii. Then I understood and coded the parsing of the csv file input
        iv. Then I figured out how to measure nondeterminism creating the print_nondeterminism function
        v. Then I did a checklist of the things that needed to be done for when the branches accepted and I proceeded to code them

   vi. Then I figured out specifically how to trace the path using the tree structure
   vii. Then I parsed the path trace output
   viii. Then I did the reject output figuring out how to find the longest path
   ix. Then I unified the code and added all necessary comments to explain it better

b. Management of testing
   i. I used the a plus testing file provided by Professor Kogge and did some examples of when the machine should reject and when it should accept by hand
   ii. I asked a friend to check my handwritten examples
   iii. I used Professor Kogge's testing files as input and checked that the output of the code matched the one of my handwritten examples.

12. Did you do any extra programs, or attempted any extra test cases - No