CSE 30151 Theory of Computing
Fall 2017
Project 2-Finite Automata

Version 2

1 Overview

The goal of this project is to have each student understand at a deep level the functioning of a **finite automaton** (FA), how "programs" for such a machine should be written, and what we mean by "complexity" of such programs when executed on real inputs. This design should set the stage for later projects on alternative classes of automaton.

The difference from Version 1 is the removal of the second to last paragraph in 2.1 and addition of *itertools* as an acceptable Python library.

2 Project Structure

Completion of this project will involve two parts. First is writing and running a program that, when given one file that represents a set of possibly many input strings and another that represents a rule set for a deterministic finite automaton (DFA), computes the states that the DFA transitions through for each input string, and provides execution statistics.

Second is writing a program that takes the rule set for a NFA and converts it into a rule set for an equivalent DFA. This latter output should be compatible with the first program so that it can be run with some string as input, and produce a correct accept/reject.

We are not after world-class performance here, just some relatively simple code that is functional and from which you can draw insight. You are free to go beyond the minimal requirements and produce enhanced code, and if in the instructor's view it truly represents valuable extensions, then it will be considered for extra credit.

You are free to use C, C++, or Python. The latter in particular has proven in the past as perhaps the most efficient way to get decent working code (the overall goal). If you use Python, feel free to use the *time*, *csv*, *sys*, and *copy* modules. Use of any other modules requires preapproval of the instructor.

2.1 DFA

This program, to be called **dfa-*team***, where *team* is the name of your team, must be written by each group, and must take the following inputs (in this order):

- An input file defining the FA to be simulated

- An input file providing multiple character strings to run one at a time against the machine.

The input file containing the FA's program should be read in and internally processed to whatever representation the group is using. As each rule is read in, an integer "Rule #" should be associated with it, assigned in sequential order. The Rule #, followed by a ":", and an echo of the rule should be dumped to stdout so that later traces can be followed.

After reading in the program file, and before each character string is processed from the second file, the FA should be reset to its start state, and all statistics should be cleared. The name of the test file shall be echoed to stdout. Then the FA should be allowed to run against the next line of text from the input file until one of:

- There are no more input characters in the current string,
- The input character is not part of the alphabet,
- There is no rule covering the current state and input character.

The output from each run is documented in Section 4.3.

After processing of each string, if there are more input strings (as signified by more lines in the input file), the FA is reset and execution repeated on the next string.

2.2 NFA-to-DFA Translator

This program, to be called **nfa2dfa-*team***, where *team* is the name of your team, should read in a machine file in the same format as for the DFA, and produce an output file that represents a DFA version of the original input. It is expected that this original input file is for an NFA, that is, there may be either multiple transitions from the same state and same input symbol, and/or transitions with ϵ as the character.

2.3 Test Files

Under the Projects tab of the class website https://www3.nd.edu/ kogge/courses/cse30151-fa17/index.html there is a directory called Project2. Within this directory there are several test files associated with **dfa-*team***:

- Four different machine definition: M1.txt, M2.txt, M3.txt, Mystery.txt
- For the first three machines, sets of strings that should be accepted or rejected by the associated machines
- For the Mystery machine, a set of strings whose acceptance is unknown

2.4 Student-supplied Machines

In addition to instructor-supplied machines and test program, each individual student shall create their own machine and test strings for each of the dfa and nfa cases. These should be documented in the readme and included with the code. If there are 3 students in a group, then 3 different machines and test code shall be included. The name field of each machine should thus include the *netid* of the student doing the design.

3 Documentation

Two pieces of documentation, both in PDF format, are required. One (entitled **readme-*team*.pdf** is submitted once by the entire team; the other **teamwork-*netid*.pdf** is submitted separately by each member of the team.

3.1 readme-team

A key part of what your teams submit is a **readme-*team*.pdf** that includes the following in this order:

1. The members of the team.
2. Approximately how much time was spent in total on the project, and how much by each student.
3. A description of how you managed the code development and testing. Use of github in particular is a particularly strong suggestion to simplifying this process, as is some sort of organized code review.
4. The language you used, and a list of libraries you invoked.
5. A description of the key data structures you used, especially for the internal representation of states and the state machines and the transitions.
6. For each of the NFA (N) and its equivalent DFA (D), plot the number of states of D v. the number of states of N.
7. If you did any extra programs, or attempted any extra test cases, describe them separately.

Only one member of the team need submit this report to their dropbox.

3.2 teamwork-netid

In addition, each team member should prepare a brief discussion of their own personal view of the team dynamics, and stored in a PDF called **teamwork-*netid*.pdf**, where netid is your netid. The contents should include:

1. Who were the other team members.
2. Under whose netid is the **readme-*team*.pdf**, code, and other material saved.
3. How much time did you personally spend on the project, and what did you do?
4. Which machines did you design and how did you generate test strings?
5. What did you personally learn from the project, both about the topic, above programming and code development techniques, and about algorithms.
6. In your own words, how did the team dynamics work? What could be improved? (e.g. did you use github and if so did it help, did you meet frequently enough, etc.)

7. From your own perspective, what was the role of each team member, and did any member exceed expectations, or vice versa.

Each student's submission here should be in their own words and SHOULD NOT be a copy of any other team members submission, nor should they be shared with the other team members. These reports will be kept private by the graders and instructor, and will be used to ensure healthy team dynamics. The instructor retains the right to adjust the score of an individual team member from the base score (both up and down) on the basis of these reports. Also, a composite of all such reports from all projects will be used to create an overall "lessons learned" at the end of the project in what techniques seemed to work better, and where problems arose. These hopefully will be of use for the next project.

4 File Formats

4.1 Machine Format

The file provided with the rules for the FA shall consist of a file where each line provides distinct information:

- Line 1: The name of the FA program. This should be echoed to stdout before the rules are echoed.
- Line 2: The alphabet to be used: only single ASCII letters are allowed, comma separated, as in a,b,c,...

  Any ASCII character other than $\sim$ is allowed. That character is reserved to stand for ϵ in transition rules for NFAs.
- Line 3: The names of the states, separated by commas, as in q0,q1,... There is no constraint on the length or character set of a state name.
- Line 4: The name of the state that should be considered the start state.
- Line 5: A comma separated list of state names that should be marked as accepting states.
- Line 6 and beyond: one transition rule per line, in a comma-separated format:

  $Initial\_State\_Name, Input\_Symbol, New\_State\_Name$

  The *Input_Symbol* is any symbol from the defined Σ from line 2, or (when describing an NFA) optionally a $\sim$ that stands for an ϵ.

Processing of rules stops with the end of file.

The comma-separated format should be compatible with a ".csv" form, allowing a machine description to be prepared easily by a spreadsheet such as excel if desired. In fact a valid extension for such files could be ".csv."

Note that it may be that such files are produced on a Windows machine that adds a carriage return and a linefeed to the end of each line.

4.2 Test File Format

Each test file is a text file, with one line for each string to be input to the machine. Note that it may be that such files are produced on a Windows machine that adds a carriage return and a linefeed to the end of each line.

4.3 Output Format

After processing the rules file, the output from each string run shall be sent to stdout and consist of:

- The input string being processed, prefixed by "String: "
- For each transition, the following on a separate line:
- $Step\_number, Rule\_Number, Initial\_State\_Name, Input\_Symbol, New\_State\_Name$
- After all transitions have been performed, print either "Accepted" (if the last state was an accepting state) or "Rejected" (otherwise)

The comma-separated format should be compatible with a ".csv" form, allowing a redirect to a ".csv" file so that the output can be read by a spreadsheet program such as excel.

5 Submission

- Each team member should have in their own Sakai directory for this course a directory called Project2, where all submissions should go.
- When the team is ready to submit, one (and only one) team member will place copies of all code at output in the designated common directory.
  - Your code should be runnable on any of the studentnn.cse.nd.edu machines so that if there is an issue the graders can run the code themselves.
  - If you wrote in a compiled language like C++, include all needed source files (excepting standard libraries), a make file, and a compiled executable. The source code is there to allow the graders to look at the code for comments and to resolve any discrepancies that may arise in looking at your results.
  - If you wrote in a language like Python make sure your code is compatible with one of the versions supported on the studentnn.cse.nd.edu machines, again to allow the graders to check something if there is an issue.
- Also in the same directory as the code the team should place an output file for each of the test files that you ran. The format should be as described in Section 4.3, and the name should be the same name as the test file but with a ".csv" rather than a ".cnf" extension. For this project there should be 5 of them.

In addition, every team member should include in their own Project2 Sakai directory:

- A copy of their **readme-*team*** file, again in pdf. For the team members who did not upload code and readme's

- Machine and test files for the DFA and NFA designs that the student did themselves.
- The output files from running the above machine files against the string files.
- A short readme describing what the machines are, what language they were supposed to accept, and whether or not the test set showed proper operation.

6 Grading

Grading of the project will be based on a 100 points, divided up as the following

- Points off for late submissions (10 points per day).
- 5 points for following naming and submission conventions.
- 10 points for "reasonably" commented source code.
- 50 points: 25 points each based on the percent of cases you got correct for running **dfa** against the provided test cases, and 25 points for similarly running **nfa2dfa** through *dfa* against the provided test cases.
- 20 points for completeness and quality of the readme file.
- 10 points for the student-designed machines and their tests.
- 5 points for the teamwork report.

All but the last two items will be common to all members of a team. The last two are specific to each member.

Special consideration will be given for successful projects done by 1 or 2 person teams, and/or for exceptional algorithm performance (as judged by the instructor).