CS-4700/5700 Automata, Computability, and Formal Languages
PJ01.docx

PJ01 – Finite Automaton Simulator

In this project you will write a program that will read a simplified description of a finite automaton, validate it, and then simulate it on each string read from an input text file. Each string that is accepted by the DFA will be echoed to a text file; in addition each machine will have a log file prepared containing specific pieces of information.

Undergraduate students are only responsible for being able to process deterministic machine descriptions, while graduate students must also be able to process nondeterministic machine descriptions. However, all students will receive the same set of machines, so undergraduates will need to be able to determine whether a machine is deterministic or not.

To avoid operating-specific case-sensitivity issues, all filenames will be all lowercase. So that all of the files can be in the same directory and the following naming conventions will be used:

Base name: mxx where xx is a two digit number. The first machine will have a base name of m00 and the remaining machines will be numbered sequentially. This should allow you to write a wrapper program that processes all of the machines in a single run.

Machine description file: basename.fa

Accepted strings: basename.txt

Log file: basename.log

Input file: strings.txt (the same file for all machines)

Machine Description File – basename.fa

All machine descriptions have the following common characteristics:

- The maximum number of states in the machine is 256.
- The states need not be numbered sequentially.
- The start state is state 0.
- State 255 is an inferred non-accepting trap state.
- The input alphabet consists of all of the printing characters (as defined by the C standard library function isprint()) except the grave accent (a.k.a. back tick) character (`), which serves as an 'epsilon' character. Note that the ASCII code for the back tick is 96 (0x60). Also note that the space character (ASCII code 32 (0x20)) IS a printing character.
- Transitions to State 255 may or may not be included in the transition rules contained in the description file.

- Not all transitions may be included – missing transitions are to be assumed to go to the trap state.
- The nominal alphabet for the machine consists of those characters for which at least one transition rule is defined.

With this common framework, the description file contents are as follows:

Line #1: Curly-brace enclosed, comma-separated list of accept states.

Line #2+: One transition rule per line in the form: state, symbol, new_state

Note that a space character is encoded as a literal, meaning that the transition rule will contain a state number, a comma, a space, a comma, a state number.

A nondeterministic machine description may have one or both of the following traits that a deterministic machine does not:

- Multiple transition rules for the same state/symbol combination.
- Epsilon transition rules.

Upon reading the machine description file, your program should validate it. There are three possible outcomes from this process: DFA, NFA, INVALID. Since the states and alphabet are inferred from the transitions, there are only a limited number of things that can be checked.

- First, the list of accept states should consist of zero or more valid states (namely non-negative integers strictly less than 255).
- Second, the transition rules should only be defined for valid alphabet symbols (i.e., printing characters) or, in the case of an NFA, eplison.
- Third, the transition rules should only be defined from and to valid states.

If there are no problems with the format, then the machine is a DFA provided it has no eplison transitions and if no state has more than one transition rule for the same symbol. Should a machine have the same transition rule repeated, you may choose to ignore all but one of them, or you may classify the machine as an NFA, or you may declare the machine as invalid. This is so that you can essentially let your program deal with this situation in whatever way is natural for it – you can take away from this that such a machine description file will not be included deliberately (but past machine description files have done so inadvertently).

Undergraduates are required to properly identify NFA descriptions, but do not need to simulate them.

CS-4700/5700 Automata, Computability, and Formal Languages
PJ01.docx

String Input File – strings.txt

There will be a single input file, named strings.txt, that will be run against each machine. This is a simple 8-bit ASCII text file (NOT Unicode). Each line contained in the file is a separate string. The terminating end-of-line (EOL) sequence is NOT part of the string. The EOL sequence could use either the DOS/Windows convention (CRLF) or the UNIX convention (LF), your program must be able to process either.

There is no limit on the length of a string. Blank lines are the empty string. ALL strings are EOL-terminated, including the last one in the file.

Output File – basename.txt

The strings that are accepted should be echoed to the output file and should be identical to the input string – meaning in particular no missing/extra spaces before or after the string – and should be terminated with an EOL terminator (either CRLF or just LF – you may use whatever your system produces naturally) hence, when opened in a text editor, the cursor should be able to go to the beginning of the line after the last string, but this line should be empty. An empty string is therefore a line containing only and EOL terminator.

Log File – basename.log

After reading the machine description file and processing the input file, the program should log the following information to an ASCII text file (in the following order, one item per line).

Valid: DFA | NFA | INVALID

States: n

This number should include all states that have explicit transitions to/from them plus state 255 only if it has any transitions to it from a used state for any member of the alphabet.

Alphabet: abcd

The nominal machine alphabet, in ASCII order, without delimiters or escape characters of any kind. Note that exactly one space follows the colon after the label. This space is NOT part of the alphabet. Also note that epsilon is NEVER an element in the alphabet!

Accepted Strings: a / m (where a = number accepted; m p number of strings in input file)

The last lines should be “0 / 0” for any invalid machine (no need to even attempt running the machine) or, for undergraduates, for NFA machines unless you choose to support them.

CS-4700/5700 Automata, Computability, and Formal Languages
PJ01.docx

SUBMISSION

You need to submit a single ZIP file. At the top level should be a single-spaced report (in PDF format) that is no more than five pages long (two to three is preferred). This report should focus on describing the technical approach taken in implementing the simulator. Included in this description should be a short section indicating the programming language and/or development environment that you used and any special configuration settings needed by someone that would like to reproduce your program as well as how to run your program.

There should also be two folders at the top level. The first is named "results" and should contain your output and log files for all of the machines. It does not need to have any other files, including the machine description files or the input file. There should also be a folder named "code" that contains all of the source code for your program. Included should be all files needed by someone to reproduce your program assuming they have the appropriate development environment. This folder can contain subfolders as appropriate.

Keep in mind that the grader is very unlikely to even attempt to run your program – they likely will not know the language you used, let alone have the necessary tools. Thus it is important that your report file describe your approach and algorithm very clearly AND that your code be well organized and commented – the grader WILL review your code.

GRADING CRITERIA

Report/Code: 50%

- 70% Technical merit – algorithm is well explained.
- 10% Programming environment details well explained.
- 10% Code is well organized
- 10% Code is well documented

Machines: 50%

Each machine will count equally toward this portion of the grade.

For EACH machine:

- -20% for mischaracterizing the machine type.
- -5% for each state different than the correct number of states
- -5% for each nominal alphabet symbol in error (either missing or extra)
- -1% for each string acceptance error (either falsely accepted or falsely rejected)

If the alphabet or strings are not in the correct order, this can result in a very low score.