Final Project User Manual

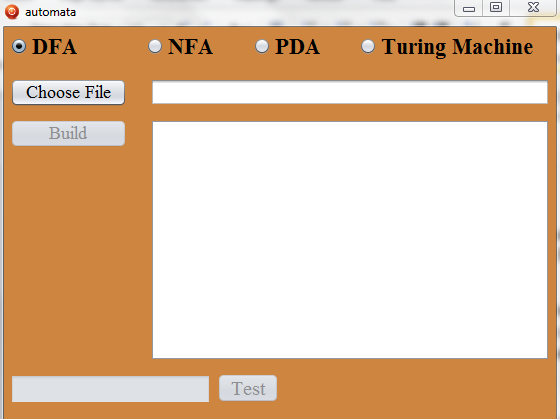
How this program works is that first, there will be an external text file with the configuration of the machine of your choice. Then you will select that file and load it into the program. The program then will build the machine and then test it with input strings.

\*DFA and NFA will only work with binary strings or ‘e’ for empty string

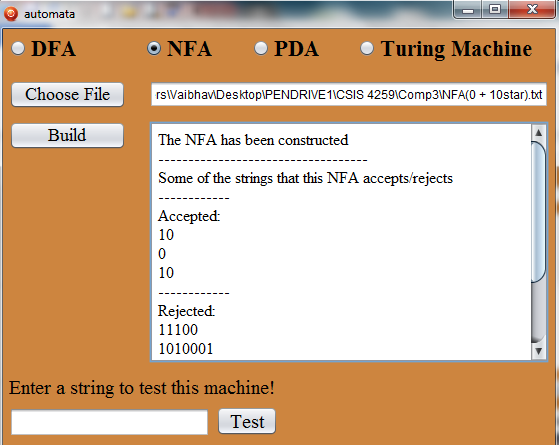
\*PDA will work with binary string and ‘(‘ ‘)’

\*TM will work with {0,1,2} ‘(‘ ‘)’ and & (empty string or blank symbol)

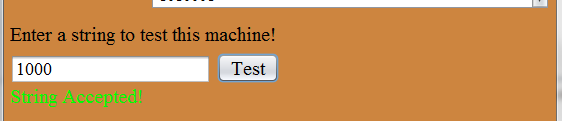
Overview of the program

1- Go ahead and open automata.jar which is an executable file

2- Select the machine of your choice and then click ‘Choose File’ to select the appropriate machine configuration file, then ‘Build’ which will build the machine programmatically



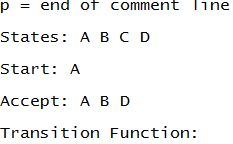
`

3- Now you can input strings to test this machine

Overview of the configuration files

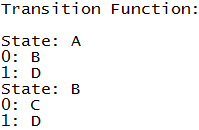
So how this program builds the machine is from reading an external .txt file which will hold all of the configuration info of the appropriate machine. For convenience, we have provided several pre-built configuration files (examples) for DFA (file name starts with ‘DFA’ followed by the machine language), NFA (file name starts with ‘NFA’ followed by the machine language), PDA (file name starts with ‘PDA’ followed by the machine language) and Turing Machine (file name starts with ‘TM’ followed by the machine language). These examples range from very simples machines to complex machines. Alternatively you can set up your own configuration for any language you want; use the file that has the name ‘createyourown’ for the appropriate machine.

Now file details and formatting: All of the files will have lines that begin with ‘c’ which indicates that it is a comment line. You should read those because they will have important info about the particular machine. ‘p’ means it is the end of comment lines. Now when it comes to formatting, you have to be very careful if creating your own machine. When the program starts reading the file, it will ignore all of the ‘c’ and ‘p’ lines. Then it will start looking for certain keywords at certain positions. If not found, either the program will throw an exception or it will not build the machine properly. So you have to be very careful about spacing and other key words. All of the files will have the following section. Where “States:” represents the set of states, “Start:” represents the start state, “Accept:” represents the accept state(s) and then the transition fucntion will go below “Transition Function:”. Take note of all of the spacings.



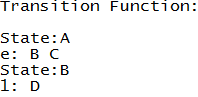
The transition function will differ for different machine types.

Info on DFA Transition Function: The following is the transition function for DFA files.



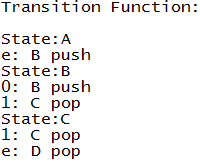
This transition function indicates for state A, ‘0’ will take you to state B and ‘1’ will take you to state D. For state B, ‘0’ will take you to state C and ‘1’ will take you to state D. (others are not shown here). Remember the space after “State:” and after “0:” and “1:”

Info on NFA Transition Function: The following is the transition function for NFA files. ‘e’ for ebsilon transition



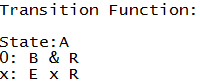
This function indicates for state A, ‘e’ transition will automatically take you to state B and state C. This state doesn’t have ‘0’ or ‘1’ transition. For state B, ‘1’ will take you to state D. this state doesn’t have any ‘0’ or ‘e’ transition. Take note that unlike the DFA file, you can’t have a space after “State:” for NFA, PDA and TM files. One more thing, the order of putting the transition for ‘0’, ‘1’ and ‘e’ matters. For example if you have a state where you have ‘0’ ‘1’ and ‘e’ transitions, you must put ‘0’ transition first, then ‘1’ and then ‘e’.

Info on PDA Transition Function: The following is the transition function for PDA files. Only ‘x’ will be pushed/popped from stack. PDAs will not work with ebsilon transitions. ‘e’ here just means that before reading a symbol, push ‘&’ (which is delta) on stack and then read and then when end of string is reached and the top of stack is ‘&’, then pop it and accept, else reject.



This function indicates for state A which is the start state, automatically push ‘&’ on the stack and move to state B. For state B, ‘0’ will take you to state B and push ‘x’ on stack and ‘1’ will take you to state C and pop ‘x’ from stack. For state C, ‘1’ will take you to state C and pop ‘x’ and if end of string is reached and top of stack is ‘&’ then accept.

Info on TM Transition Function: The following is the transition function for TM files. ‘&’ represents the blank symbol. The only other extra symbol to write on the tape (other than the blank symbol) is ‘x’. ‘R’ means move head right, ‘L’ means move head left and ‘S’ means stay.



This function indicates for state A, ‘0’ will take you to state B, write ‘&’ on the tape and move the head right. This is the same as ȡ(A,0) = (B,&,R). On reading an ‘x’, the machine changes to state E, write ‘x’ and move head to right.

Code Overview:

Written in Java

Lines of code: 2,000+

Total class files: 10

Classes: State.java, NFAState.java, PDAState.java, TMState.java, InputFileException.java, Demo.java, DFA.java (constructs the dfa from input file and processes the string on dfa), NFA.java (constructs the nfa form input file and processes the string on nfa), PDA.java (constructs the pda from input file and processes the string on pda), TuringMachine.java (constructs the tm from input file and processes the string on tm),

Folder Content: several examples of different machines, automata.jar, src folder which holds all of the source code.