Practice lessons of Theory of automata and Formal languages

Prof. Karl Thurnhofer Hemsi
Computer Science and Computer Language Department
University of Málaga

October 19, 2022

Practice 2

Automata in JFLAP

2.1 JFLAP

JFLAP is a software implemented in JAVA to experiment with formal language topics that include deterministic (DFA) and non-deterministic finite automaton (NFA), pushdown automaton (PDA), multi-tape Turing machines, various types of grammars, syntactic analysis, etc.

- It has the ability to simulate input strings in non-deterministic automata, build syntactic analysis tables and analyze trees for grammars. The automata represented in JFLAP are finite automata, pushdown automata and multitasking Turing machines.

- Allows conversion between equivalent structures: a wide range of conversions are available, for example, regular expression to DFA, non-deterministic FA to deterministic FA, NPDA to grammar, grammar to NPDA, DFA to minimal DFA, free context grammars to grammars in the Chomsky normal form, etc.

In this practice lesson we will learn to use this tool and see its usefulness:

- Download the JFLAP program and the practice files from the Virtual Campus.

- Run the JFLAP.jar file using 'Right button → Run with Java Runtime Environment' or the command `java -jar JFLAP.jar`

2.2 Finite automaton (FA)

Introduction

1. Select the option 'Finite Automaton'.
2. Open the file "example_intro.jff".
3. Select the option 'Input → Step with closure' and enter the string *aabbaba*.
4. Press the 'Step' button and see what happens.

Part I

5. Open the file "DFA_cxc.jff".
6. What type of automaton is shown? Why?

7. Test the following chains using the option 'Input → Multiple Run': *aac, cac, ccc, caabbbc, caaab, acccbbbc.*

8. What language does this DFA recognize?

Part II

9. Open the files "NFA_aa.jff" and "NFA_abstar.jff".

10. What types of automaton are they?

11. Test the chains: *aa, ab, abbba, abbaa, a.*

12. What languages do each of them recognize?

13. Let L_1 and L_2 be the languages generated by "NFA_aa.jff" and "NFA_abstar.jff" respectively. From them we want to generate an automaton that recognizes L_1L_2. Use the option 'Convert → Combine automata' and then include the missing transitions.

14. Open the file "NFA_abstar_aa.jff". Does this automaton recognize the same chains as the previous one? Use the option 'Input → Multiple Run' to test some strings, and explain your answer.

2.3 Regular Pumping Condition

In JFLAP a turn-based game is proposed against the computer, in which player 1 tries to find a decomposition that is always in the language (even if it is "pumped"), while player 2 tries to avoid it. If this is achieved, it means that there is no adequate decomposition and the language will not be regular.

- Player 1 chooses a value m (equivalent to the n seen in class).
- Player 2 chooses a string $w \in L$ with $|w| \geq m$.
- Player 1 chooses the decomposition $w = xyz$ that meet the second and third pumping conditions (in class it was seen as $x = uvw$)

- Player 2 tries to find a value $i \geq 0$ (in class it was called m) such that the fourth pumping condition doesn't come true. In this way, if he does, player 2 wins, but if he does not find it, player 1 wins.

15. Select the option 'Regular Pumping Lemma'.

16. Select the option 'Computer goes first' (that is, you are player 2), and select the third example: $L = \{ww^R : w \in \{a, b\}^*\}$.

17. Follow the instructions and beat the computer.

18. For the string that appears to you, will it be possible to find a decomposition such that when you "pump" it then it belongs to the language? Why?

19. Make the computer be player 2, and try not to win.

2.4 Non-deterministic Pushdown Automaton

In JFLAP the NPDA are represented in a similar way as the FA, but the transitions are formed by triples in which the first element is the chain to consume, the second is

the chain that we take out of the stack, and the third is the chain that we put in the stack. In JFLAP, the stack is always initialized with the symbol Z, and the way it has to accept strings can only be, or reaching a final state, or that the stack is empty (not both as the definition seen in class).

20. Select the option 'Pushdown automata' and select the option 'Single Character Input'.

21. Open the file "ExampleAPND.jff".

22. Select the option 'Input → Step with closure' and enter the string *aaaabbbb*. As acceptance mode, select the option 'Empty Stack'. What happen?

23. Enter any other string and study the behavior.

2.5 Minimization of DFA

A specific language L can be represented by an DFA, but it may not be the optimal automaton to define this language, that is, it might have non necessary nodes in order

to generate the language L. For those case, there exists a minimization algorithm that allow us to generate the minimal DFA from a given one. In this section we are going to see how JFLAP performs this procedure.

24. Select the option 'Finite Automaton'.

25. Open the file "dfaToMinDFA.jff".

26. Select the option 'Convert $\rightarrow$ Minimize DFA'.

    A new window will appear on the right panel with a state tree consisting of three nodes: a root node with all the statuses of the automata, and two child nodes that divide the set of states into endings and not endings states. The minimization procedure consists of dividing the states into more leaf nodes by making groupings based on the arrival of transitions of a certain terminal symbol.

27. Select the node labeled "Nonfinal", and press the 'Set Terminal' button. Enter the symbol b. What has happened? Then, repeat the process with the symbol a. What happened?

For b, no division happens because there is no transition that goes from the group that we are dividing to the other group (q_6); for a yes, since there are transitions arriving from q_0 and q_4, then these two states must belong to a group.

28. Select one of the nodes and indicate in the left panel the states that should be in that node. Do the same with the other node. When you finish, click on 'Check Node' to verify that the partition is correct.

29. This procedure can be performed automatically. Press the 'Auto Partition' button to perform only the partition of the last level of the tree. If we want to do all partitions at once (recursively), click on 'Complete Subtree'.

30. To finish, press 'Finish'. The status of the new minimal automaton appears in the right panel.

31. Finally, we must complete the transitions based on the original automaton. We can also use the 'Hint' buttons to add transitions one by one, or 'Complete' to add them all. Finally, we check with 'Done?'

Activities

1. Consider the language over the alphabet $\{a, b\}$ that only contains the string a.

    a. Build a DFA that recognizes this language and rejects all those strings that do not belong to the language.

    b. Test the automaton that you have created by introducing 6 chains.

2. Finite automaton in Octave:

    a. Open the Octave `finiteautomata.m` script and test it with the given example (see script help) in the GitHub repository.

    b. Specify in `finiteautomata.json` the automaton created in Activity 1 and test it with the script!

3. Test the Free Context Pumping Condition for the first three examples.

4. Build an NPDA that recognizes the language $L = \{0^n 1^{2n} : n > 0\}$.