Formal Languages and Automata Theory
List of Projects

Note: The list of projects can be updated during the semester: more projects can be added.

Students who decide to work on a project must announce the lecturer on the chosen topic. The project consists of a program solving the problem as well as documentation. The documentation should contain the problem what is solved, examples, design choices of the implementation and a small user manual of your tool.
You could obtain maximum 2 points for the project. The projects should be uploaded and presented before the first examination.

Project 1 *Simulation of an DFA.* A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set called the states, Σ is a finite set called the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subset Q$ is the set of accept states. We design a DFA and give a string as input. The program should check the validity of that string and displays the states that is encountered by the input string.

Inputs to the program:

1. The number of states that the DFA will contain.
2. Here we take the Σ the set of alphabets constant as $0, 1$ they might be altered or increased changing the course of the program.
3. Now we define the next states for every state for both alphabets 0 and 1; and if a state is final state or not.
4. $q_0 \in Q$ the initial state is taken as input and we have our DFA ready.
5. Next we take one string after another and check the validity of that string and the program shoes the number of states the machine traversed for that input string.

Program output:

1. Acceptance of the input string.
2. States traversed by the machine for that input string.

Project 2 Construction of an DFA equivalent to a given regular expression.

Project 3 DFA minimization (see lecture notes and Lecture 14 from Kozen book available on the website).

Project 4 Write a program that reads a deterministic finite automaton from a file and, using (f)lex (http://flex.sourceforge.net/), (a) tests whether the automaton is deterministic, (b) tests whether the language used by the automaton is void, and if not finds a word accepted by the language, (c) simulates the automaton on the given word (printing the transitions/states)[1].

[1]From Gabriel Istrate lecture notes

Project 5 Consider the following puzzle: "On one side of a river are three humans, one big monkey, two small monkeys, and one boat. Each of the humans and the big monkey are strong enough to row the boat. The boat can fit one or two bodies (regardless of size). If at any time at either side of the river the monkeys outnumber the humans, the monkeys will eat the humans. How do you get everyone on the other side of the river alive?" Show that the language of solutions to the puzzle is regular. Write a finite automaton for the puzzle to a file (perhaps using a script if you need it). Using the previously written program find and print a solution to the puzzle. The printing should be done to be "understandable by humans"[2].

Project 6 Conversion to Chomsky normal form.

Project 7 Simulation of deterministic pushdown automata.

Project 8 Applications with regular expressions. Consider the Facebook metrics available at `http://archive.ics.uci.edu/ml/datasets/Facebook+metrics#`. Use regular expressions for computing certain statistics. You can use the paper mentioned at the link for some interesting ones.

Project 9 Applications with regular expressions. Consider the Amazon books reviews metrics available at `http://archive.ics.uci.edu/ml/datasets/Amazon+book+reviews`. Use regular expressions for computing certain statistics, for example the two statistics from `http://ataspinar.com/2016/01/21/sentiment-analysis-with-bag-of-words/`, section Data Collection.

Project 10 (1) Given a grammar G, specify its type.

(2) Given the grammars G_1, ..., G_n, construct the grammars which generate $L(G_1) \cup ... \cup L(G_n)$ (union), $L(G_1) \cdot ... \cdot L(G_n)$ (product), $L(G_1)^*$, ..., $L(G_n)^*$ (Kleene closure) (you should use the algorithm developed at (1)).

[2]From Gabriel Istrate lecture notes