Automata, Computability, and Formal Languages
Fall 2023, Final Exam

Problem 1. Finite automata and regular languages.

Problem 1a. Design a finite automaton for recognizing non-empty words that start with the letter a. Assume that the input strings contain only symbols a and p. The easiest is to have three states:

- the starting state s corresponding to the empty input string,
- the state f indicating that the first symbol that we have read was a, and
- the state e indicating all other words.

You just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the word *app*.

Problem 1b. Explain why in most computers binary numbers are represented starting with the lowest possible digit.

Problem 1c. On the example of the above automaton, show how the word *app* can be represented as xyz in accordance with the pumping lemma.

Problem 1d. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a. (If you are running out of time, it is Ok not to finish, just eliminate the first state.)

Problem 1e-f. The resulting language can be described by a regular expression $a(a \cup p)^*$. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.

Problem 2. Beyond finite automata: pushdown automata and context-free grammars

Problem 2a. Suppose that the university received a big gift of calculators to be distributed between students from different departments, on the condition that they should be distributed fairly – i.e., proportionally to number of student in each department. Suppose that Computer Science department (C) has twice more students than Electrical Engineering (E), and Mechanical Engineering (M) has three times as many students as E. In this case, e.g., a distribution CECMMM is fair. Prove that the set of all such "fair" sequences is not regular and therefore, cannot be recognized by a finite automaton.

Problem 2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word *app*.

Problem 2c. For the context-free grammar from the Problem 2b, show how the word *app* can be represented as $uvxyz$ in accordance with the pumping lemma.

Problem 2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

Problem 2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word *app*.

Problem 2f. Use the general stack-based algorithms to show:

- how the compiler will transform a Java expression 14/7/3 into inverse Polish (postfix) notation, and
- how it will compute the value of this expression.

Problem 3. Beyond pushdown automata: Turing machines

Problem 3a. Prove that the language of fair sequences as described in Problem 2a is not *context-free* and therefore, cannot be recognized by a pushdown automaton.

Problem 3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word *app*. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

Problem 3d-e. Design Turing machines for computing $a - 1$ in unary and in binary codes. Trace both Turing machines for $a = 5$.

Problem 4. Beyond Turing machines: computability

Problem 4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

Problem 4b. Prove that the halting problem is not algorithmically solvable.

Problem 4c. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give a current formal definition of feasibility and give two examples:

- an example of an algorithm's running time which is feasible according to the current definition but not practically feasible, and
- an example of an algorithm's running time which is practically feasible but not feasible according to the current definition.

Problem 4d. Briefly describe what is P, what is NP, what is NP-hard, and what is NP-complete. Is P equal to NP?

Problem 4e. Give an example of an NP-complete problem: what is given, and what we want to find.

Problem 4f. Give definitions of a recursive (decidable) language and of a recursively enumerable (Turing-recognizable) language.