Test 2, Theory of Computation, Spring 2026

Problem 1. Prove that it is not possible, given a program that always halts, to check whether this program always computes $2^n \,\%\, n$. *Comment:* here, as usual, $a \,\%\, b$ means remainder.

Problem 2. Design a Turing machine that computes $n - 4$ in binary code for all $n \geq 4$. Trace this machine on the example of $n = 1101_2$.

Problem 3. Use a general algorithm for a Turing machine that represents composition to transform your design from Problem 2 into a Turing machine for computing $f(f(n)) = n - 8$.

Problem 4. Give a formal definition of feasibility and explain what is practically feasible. Give two examples:

- an example when an algorithm is feasible in the sense of the formal definition but not practically feasible, and
- an example when an algorithm is practically feasible, but not feasible according to the formal definition.

These examples must be different from the examples that we had in class, in posted lectures, homeworks, or in last years' solutions.

Problem 5. What is P? NP? NP-hard? NP-complete? Brief definitions are OK. What do we gain and what do we lose when we prove that a problem is NP-complete? Explain one negative consequence (what we cannot do) and one positive one (what we can do).

Problem 6. What is propositional satisfiability? Give an example. Explain why this problem is important in software testing.