Theory of Computation, Spring 2025
Solutions to Test 3

Problem 1. Explain where and how, in proof that satisfiability is NP-hard, we use the two physical assumptions: that all speeds are bounded by the speed of light, and that the volume of a sphere is proportional to the cube of its radius.

Solution. These two assumptions are used in two places:

- to prove that the number of neighbors – that can affect the state at the next moment of time $t + \Delta t$ – is bounded by a constant that does not depend on the size of the input, and
- to prove that the overall size of the resulting CNF formula is polynomial.

In the first place, all processors whose state at moment t can affect the state of the current cell at moment $t + \Delta t$ are located inside a sphere of radios $r = c \cdot \Delta t$ with center in this cell. The volume of this sphere is $v = (4/3) \cdot \pi \cdot r^3$, so within sphere, we can have no more than $v/\Delta v$ processors. This bounds does not depend on the size of the input, so indeed, the number of neighbors is bounded by a constant that does not depend on the size of the input.

In the second place, we are translating an algorithm $C(x, y)$ that finishes computations in polynomial time T. During this time T, only cells located at distance $\le c \cdot T$ can affect the computation result. All these cells are located in the sphere of radius $R = c \cdot T$. The volume V of this sphere is $\text{const} \cdot R^3 = \text{const} \cdot T^3$. Thus, the number of cells N_{cells} is bounded by $V/\Delta V = \text{const} \cdot T^3$. In the CNF formula, we have formulas corresponding to all possible triples (i, b, t). There are N_{cells} cells i, B bits b, and $T/\Delta t$ moments of time t. Thus, the length of the formula is bounded by $T^3 \cdot B \cdot T \sim T^4$. We know that $T(n)$ is bounded by a polynomial of the size of the input. In this case, T^4 is also bounded by a polynomial – 4th power of the polynomial bound for T.

Problem 2. Use the general algorithm to translate the formula

$$(p \vee \neg q \vee r \vee s) \,\&\, (\neg s \vee t)$$

into 3-CNF.

Solution: According to the general algorithm, we introduce a new variable r_1 whose meaning is $p \vee \neg q$, then the satisfiability of the original formula is equivalent to the satisfiability of the formula

$$(p \vee \neg q = r_1) \,\&\, (r_1 \vee r \vee s) \,\&\, (\neg s \vee t).$$

Then, we reduce the formula $p \vee \neg q = r_1$ to CNF.

The truth table for the negation N of this formula is:

p	q	r_1	$\neg q$	$p \vee \neg q$	N
0	0	0	1	1	1
0	0	1	1	1	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	1	0	1	0

Thus, the DNF form of N is

$$(\neg p \,\&\, \neg q \,\&\, \neg r_1) \vee (\neg p \,\&\, q \,\&\, r_1) \vee (p \,\&\, \neg q \,\&\, \neg r_1) \vee (p \,\&\, q \,\&\, \neg r_1).$$

Thus, the formula $\neg p \vee q = r_1$ has the following CNF form:

$$(p \vee q \vee r_1) \,\&\, (p \vee \neg q \vee \neg r_1) \,\&\, (\neg p \vee q \vee r_1) \,\&\, (\neg p \vee \neg q \vee r_1).$$

As a result we get the following 3-CNF expression:

$$(p \vee q \vee r_1) \,\&\, (p \vee \neg q \vee \neg r_1) \,\&\, (\neg p \vee q \vee r_1) \,\&\, (\neg p \vee \neg q \vee r_1) \,\&$$

$$(r_1 \vee r \vee s) \,\&\, (\neg s \vee t).$$

Problem 3–4. Reduce the satisfiability problem for the formula

$$(a \vee b \vee \neg c) \,\&\, (a \vee \neg b)$$

to:

- 3-coloring,
- clique,
- subset sum problem, and
- interval computations.

In all these reductions, explain what will correspond to $a = T$, $b = F$, and $c = T$.

Solution: For *3-coloring*, we do the following:

- We start with the palette, i.e., with vertices T, F, and U which are connected to each other.
- We form vertices a and $\neg a$, and we connect them to each other and to U.
- We form vertices b and $\neg b$, and we connect them to each other and to U.
- We form vertices c and $\neg c$, and we connect them to each other and to U.
- We form a new vertex $a \vee b$ and connect it to U.
- We form new vertices $(a \vee b)_1$ and $\neg c_1$, connect them to each other and to T, connect $(a \vee b)_1$ to $a \vee b$, and connect $\neg c_1$ to $\neg c$.
- We form new vertices a_1 and b_1, connect them to each other and to $a \vee b$, connect a_1 to a, and connect b_1 to b.
- We form new vertices a_2 and $\neg b_2$, connect them to each other and to T, connect a_2 to a, and connect $\neg b_2$ to $\neg b$.

When $a = T$, $b = F$, and $c = T$, we can color the vertices in the following colors:

- The vertex T is colored T, the vertex F is colored F, and the vertex U is colored with color U.
- a is colored with color T, $\neg a$ is colored with color F.
- b is colored with color F, $\neg b$ is colored with color T.
- c is colored with color T, $\neg c$ is colored with color F.
- The vertex $a \vee b$ is colored with color T.
- The vertex $(a \vee b)_1$ is colored F, the vertex c_1 is colored U.

- About the vertices a_1 and b_1, we have a choice: one of them can be colored by F, another one by U.
- The vertex a_2 is colored F, the vertex $\neg b_2$ is colored U.

One can check that in this case, no two connected vertices have the same color.

For *clique*, we add the following vertices: a_1, b_1, $\neg c_1$, a_2, and $\neg b_2$. We then:

- connect a_1 with a_2 and $\neg b_2$;
- connect b_1 with a_2; and
- connect $\neg c_1$ with a_2 and with $\neg b_2$.

The values $a = T$, $b = F$, and $c = T$ correspond to two possible 2-cliques:

- a_1 and a_2, and
- a_1 and $\neg b_2$.

For *subset sum*, by applying the general algorithm, we get the following table:

	a	b	c	C_1	C_2
a	1	0	0	1	1
$\neg a$	1	0	0	0	0
b	0	1	0	1	0
$\neg b$	0	1	0	0	1
c	0	0	1	0	0
$\neg c$	0	0	1	1	0
C_1'	0	0	0	1	0
C_1''	0	0	0	1	0
C_2'	0	0	0	0	1
C_2''	0	0	0	0	1
S	1	1	1	3	3

The values $a = T$, $b = F$, and $c = T$ correspond to selecting coins $a = 10011$, $\neg b = 01001$, and $c = 00100$. To get the desired sum 11133, we also need to add coins $C_1' = 00010$, $C_2' = 00010$, and $C_2'' = 00001$.

For *interval computations*, we get the polynomial

$$(1 - (1 - A) \cdot (1 - B) \cdot C) \cdot (1 - (1 - A) \cdot B),$$

where A, B, and C are from the interval $[0, 1]$. This polynomial attains the value 1 when $A = 1$, $B = 0$, and $C = 1$, so its largest possible value is 1.

Problem 5. Show how to compute the "and" of 12 Boolean values in parallel if we have an unlimited number of processors and we can ignore communication time. Why do we need parallel processing in the first place? If we take communication time into account, how much time do we need to compute the "and" of n values? What is NC? Give an example of a P-complete problem.

Solution:

- At moment $t = 1$, Processor 1 computes $r_1 = x_1 \,\&\, x_2$, Processor 2 computes $r_2 = x_3 \,\&\, x_4$, Processor 3 computes $r_3 = x_5 \,\&\, x_6$, and Processor 4 computes $r_4 = x_7 \,\&\, x_8$.
- At moment $t = 2$, Processor 1 computes $r_5 = r_1 \,\&\, r_2$, Processor 2 computes $r_6 = r_3 \,\&\, r_4$, Processor 3 computes $r_7 = x_9 \,\&\, x_{10}$, and Processor 4 computes $r_8 = x_{11} \,\&\, x_{12}$. As a result, we get $r_5 = x_1 \,\&\, x_2 \,\&\, x_3 \,\&\, x_4$ and $r_6 = x_5 \,\&\, x_6 \,\&\, x_7 \,\&\, x_8$.
- At moment $t = 3$, Processor 1 computes $r_9 = r_5 \,\&\, r_6$ and Processor 2 computes $r_{10} = r_7 \,\&\, r_8$. As a result, we get the values $r_9 = x_1 \,\&\, \ldots \,\&\, x_8$ and $r_{10} = x_9 \,\&\, x_{10} \,\&\, x_{11} \,\&\, x_{12}$.
- At moment $t = 4$, Processor 1 computes the desired result $r_9 \,\&\, r_{10}$. One can easily check that this value is equal to $x_1 \,\&\, \ldots \,\&\, x_{12}$.

For this computation, we need 4 processors and 4 moments of time.

For computing the "and" of n values, sequential computations require at least $T_{\text{sequential}}(n) \geq n - 1$ steps. Since $T_{\text{parallel}}(n) \geq \text{const} \cdot (T_{\text{sequential}}(n))^{1/4}$, we thus need $T_{\text{parallel}}(n) \geq c \cdot n^{1/4}$.

NC is the class of all problems that can be computed on polynomial number of processors ($N_{\text{processors}} \leq P(n)$ for some polynomial $P(n)$ of the length n of the input) in polylog time, i.e., in time bounded by $P(\log(n))$ for some polynomial $P(n)$.

An example of P-complete problem is *linear programming*: checking whether a given set of linear inequalities $a_{i1} \cdot x_1 + \ldots + a_{in} \cdot x_n \geq b_i$, where a_{ij} and b_i are known, and x_j are unknowns, has a solution.

Problem 6. What can you say about the Kolmogorov complexity of the following string: 110110... in which 110 is repeated 7,000 times.

Solution: A possible program for producing this sequence x is as follows:

```
for(int i = 1; i <= 7000; i++)
     System.out.print('110');
```

This program has $33 + 31 = 64$ symbols, so the Kolmogorov complexity $K(x)$ of this string – which is the length of the shortest program for computing this string – is smaller than or equal to 64: $K(x) \leq 64$.

Problem 7. Suppose that we have a probabilistic algorithm that gives a correct answer half of the time. How many times do we need to repeat this algorithm to reduce probability of error to at most 5%? Give an example of a probabilistic algorithm. Explain why we need probabilistic algorithms in the first place.

Solution: The probability of error is $1 - 1/2 = 1/2$. We want the probability of error to be smaller than $5\% = 1/20$. So, we need to select the smallest number of iterations k for which

$$\frac{1}{2^k} \le \frac{1}{20},$$

i.e., equivalently, for which $2^k \ge 20$.

- For $k = 1$, we get $2^1 = 2 < 20$.
- For $k = 2$, we get $2^2 = 4 < 20$.
- For $k = 3$, we get $2^3 = 8 < 20$.
- For $k = 4$, we get $2^4 = 16 < 20$.
- For $k = 5$, we get $2^5 = 32 > 20$.

Thus, we need 5 iterations.

An example of a probabilistic algorithm is an algorithm for checking program correctness. If someone proposes a program $f(x)$ that, given a real number $x \in [0, 1]$, supposed to compute the desired expression $g(x)$, then a natural way to check whether the program is correct is run a random number generator several times and check that for all the resulting random values $r_1, \ldots, r_k$, we get $f(r_i) = g(r_i)$.

We need probabilistic algorithms since many practical problems are NP-complete, which means that we cannot have a feasible algorithm that always produced a correct answer. It is thus reasonable to try to find a feasible algorithm that produces a correct answer with some probability.

Problem 8. Use the variable-elimination algorithm for checking satisfiability of the following 2-SAT formula:

$$(a \vee \neg c) \,\&\, (b \vee c) \,\&\, (\neg a \vee \neg b) \,\&\, (c \vee \neg b).$$

Find all solutions.

Solution: Let us eliminate the variable a. Clauses containing a or $\neg a$ lead to the following inequalities:

$$c \le a, \quad a \le \neg b.$$

The condition that every lower bound must be smaller than or equal than every upper bound leads to

$$c \le \neg b.$$

This is equivalent to the clause $\neg b \vee \neg c$. So, for remaining variables b and c, we have the following clauses:

$$(b \vee c) \,\&\, (c \vee \neg b) \,\&\, (\neg b \vee \neg c).$$

Let us now eliminate the variables b. Clauses containing b or $\neg b$ lead to the following inequalities:

$$\neg c \le b, \quad b \le c, \quad b \le \neg c.$$

The condition that every lower bound must be smaller than or equal than every upper bound leads to the following inequalities:

$$\neg c \le c, \quad \neg c \le \neg c.$$

The second of these inequalities is always trivially true. The first one is only true when $c = 1$. Thus, $c = 1$. For $c = 1$, inequalities containing b lead to

$$0 \le b, \quad b \le 1, \quad b \le 0.$$

Thus, $b = 0$.

From $c \le a$, we can now conclude that $a = 1$. Thus, the only solution is $a = 1$, $b = 0$, and $c = 1$.

Problem 9. How is the "and" operation $f(x_1, x_2) = x_1 \,\&\, x_2$ represented in quantum computing? Provide a general formula and explain it on the example when x_1 is true and both x_2 and the auxiliary variable are false.

Solution: the general way to represent a function $f(x_1, \ldots, x_n)$ is as a mapping

$$|x_1, \ldots, x_n, y\rangle \to |x_1, \ldots, x_n, y \oplus f(x_1, \ldots, x_n)\rangle.$$

In our case, $n = 2$ and $f(x_1, x_2) = x_1 \,\&\, x_2$, so we have

$$|x_1, x_2, y\rangle \to |x_1, x_2, y \oplus (x_1 \,\&\, x_2)\rangle.$$

In particular, when $x_1 = 1$ and $x_2 = y = 0$, we get

$$|1, 0, 0\rangle \to |1, 0, 0 \oplus (1 \,\&\, 0)\rangle = |1, 0, 0 \oplus 0\rangle = |1, 0, 0\rangle.$$

Problem 10. Why do we need to study recursively enumerable (r.e.) sets? Is the intersection of three r.e. sets still r.e.? If yes, prove it, if no, provide a counterexample.

Solution. In most of the class, we used computing intuition, but in some cases, it is useful to also use set-theoretic intuition, e.g., the intuitive notions of union and intersection.

If we have three r.e. sets A, B, and C, this means that we have 3 algorithms that will eventually generate all the elements of each set and only then. To get the intersection:

- first, we run all 3 algorithms for 1 hour, then we take the intersection of what 3 algorithms produced;
- then, we run all three algorithms for 1 more hour; the lists, in general, increase, so we again take the intersection,
- etc.