SOLUTIONS TO INTRODUCTION TO ALGORITHMS

1. TD 2

We make the following assumptions about Turing machine:

- The head can move in three directions: $\rightarrow$ (right), $\leftarrow$ (left), $\downarrow$ (stay).
- For your convenience, you can choose that the starting position of the head is in the end or in the beginning of the input word, if not specified otherwise.
- You should specify, which of the states is the starting state, which is the accepting state, and which is the rejecting state (or, you can use the symbols $\triangle$ as accepting state, and ∇ as rejecting state)
- You can augment the alphabet. Even if the input alphabet is $\{0, 1\}$, for convenience you can use other symbols.
- A Turing machine is described by the alphabet, the set of states and the set of transitions. It is preferrable that you draw a Turing machine as a graph with vertices signifying the set of states, and arrows denoting the transitions.

You should always describe why the Turing machine you present, solves the given problem!

1.1. **Problem 1.** *Describe a Turing machine that allows to add* 1 *at the end of the word and stops in the state of acception.*

Idea. We start by describing an algorithm, and then we turn it into a more formal Turing machine.

a) Step 0. Assume that the head of the Turing machine starts at the leftmost position of the word w.
b) Step 1. Move the head of the Turing machine right, until we reach a state of the empty symbol $\square$.
c) Step 2. Write symbol 1 at the current position of the head of the Turing machine and pass into acception state.

In the final solution, each step will produce the *state* of the Turing machine.

Solution. Assume, for simplicity, that the alphabet is $\Sigma = \{0, 1\}$, i.e. the symbols of the word can be only 0 and 1.

The Turing machine will have only two states, we call them the state q_1 and the state q_2, according to the steps of the algorithm described above.

The state q_1 is the *initial state*, and the state q_2 is the *accepting state.*

The Turing machine is described by its respective transitions:

- $(q_1, 0) \mapsto (q_1, 0, \rightarrow)$
- $(q_1, 1) \mapsto (q_1, 1, \rightarrow)$
- $(q_1, \square) \mapsto (q_2, 1, \downarrow)$

1.2. **Problem 2.** *Describe a Turing machine that takes a word in the alphabet* $\{0, 1\}$ *and transforms the band in such a way that all* 1 *become* 0*, and all* 0 *become* 1*, and finally finished in the acception state.*

Idea. For this problem, we need only a single state for processing the letters of the word, which is equal to the initial state. We call this state q_0. When the head of the machine moves right, at some point it reaches the void symbol. At this moment the machine finishes in the acception state.

Date: 20th October 2018.

Solution. We have two states, q_0 is the initial state, and $q_1 = \triangle$ is the acception state. There is no rejection state. The transitions go as follows:

- $(q_0, 0) \mapsto (q_0, 1, \rightarrow)$
- $(q_0, 1) \mapsto (q_0, 0, \rightarrow)$
- $(q_0, \square) \mapsto (\triangle, \square, \downarrow)$

1.3. **Problem 3.** *Describe a Turing machine that takes a word in the alphabet* $\{0, 1\}$ *which iso considered as a number n written in binary base, and transforms it into the number* $n+1$ *written base* 2*, then finishes in the acception state.*

Idea. Suppose that we start at the position representing the weakest digit. Example: $100101\underline{0}$. If we should add 1 to 0, we just replace 0 by 1 and finish in the state of acception. Otherwise, we keep the information about the digit we carry in the memory, replace each 1 by 0 and continue keeping 1 until the digit 0 is encountered. In other words, the head of the machine should move left while the digits 1 are seen. After, we replace the digit 0 by 1 and finish in the state of acception.

Solution. We introduce the following states:

- q_0 for the initial state
- $\triangle$ for the accepting state
- q_1 for the state when we keep 1 in the memory

Additionally, we need to consider the situation when the blank symbol is encountered and the machine is in the state q_1, in this case we should write 1. If the blank symbol is encountered in the state q_0, this means that the input word is empty, then we put 1 and accept.

Example: $111 + 1 = 1000$.

These states require the following transitions:

- $(q_0, 0) \mapsto (\triangle, 1, \downarrow)$
- $(q_0, 1) \mapsto (q_1, 0, \leftarrow)$
- $(q_1, 0) \mapsto (\triangle, 1, \downarrow)$
- $(q_1, 1) \mapsto (q_1, 1, \leftarrow)$
- $(q_0, \square) \mapsto (\triangle, 1, \leftarrow)$
- $(q_1, \square) \mapsto (\triangle, 1, \leftarrow)$

1.4. **Problem 4.** *Describe a Turing machine that takes a word in the alphabet* $\{0, 1\}$ *and accepts if and only if the word is a palyndrome*

Examples: 01010 is a palyndrome, 1101 is not.

Idea. Let us start at the beginning of the word. We erase the first digit and remember it. Then, we go to the end of the word and check if the last digit is equal to the remembered digit. If they are equal, we erase the symbol and continue the process, otherwise finish in the rejection state.

This process continues until the word is of length 0 or 1, in which case we finish in the state of acception because the word is a palindrome.

Solution. Let

- q_0 is the initial state, and also corresponds to picking the leftmost bit;
- r_0, r_1 correspond to the two states when the head moves right carrying 0 or 1, respectively;
- p_{r0}, p_{r1} correspond to checking that the bit on the right is 0 or 1, respectively;
- p_ℓ correspond to moving left.

Then, the transitions are described as follows:

- $(p_\ell, 0) \mapsto (r_0, \square, \rightarrow)$: if we see the symbol 0 on the left, we remember the symbol and erase it, and go to the right until the end of the word;
- $(p_\ell, 1) \mapsto (r_1, \square, \rightarrow)$: the same situation, but the symbol is 1;
- $(p_\ell, \square) \mapsto (\triangle, \square, \downarrow)$: if the word is empty then it is a palindrome, finish in the accepting state;
- $(r_0, 0) \mapsto (r_0, 0, \rightarrow)$: skip any symbol unless it is blank;

- $(r_0, 1) \mapsto (r_0, 1, \rightarrow)$;
- $(r_1, 0) \mapsto (r_1, 0, \rightarrow)$;
- $(r_1, 1) \mapsto (r_1, 1, \rightarrow)$;
- $(r_0, \square) \mapsto (p_{r0}, \square, \leftarrow)$: if the blank symbol is met, we should return one position back and check the symbol;
- $(r_1, \square) \mapsto (p_{r1}, \square, \leftarrow)$;
- $(p_{r0}, 0) \mapsto (q_\ell, \square, \leftarrow)$: if the symbol corresponds, then erase it and move to the beginning;
- $(p_{r1}, 1) \mapsto (q_\ell, \square, \leftarrow)$;
- $(p_{r0}, 1) \mapsto (\nabla, 1, \downarrow)$: if the symbol does not correspond, finish in the rejection state;
- $(p_{r1}, 0) \mapsto (\nabla, 0, \downarrow)$;
- $(p_{r0}, \square) \mapsto (\triangle, \square, \downarrow)$: if we encounter blank symbol in the state p_{r0} or p_{r1}, the word had length 1 and it is a palindrome;
- $(p_{r1}, \square) \mapsto (\triangle, \square, \downarrow)$;
- $(p_\ell, 0) \mapsto (p_\ell, 0, \leftarrow)$: if a symbol 0 or 1 is met during moving left, continue moving left;
- $(p_\ell, 1) \mapsto (p_\ell, 1, \leftarrow)$;
- $(p_\ell, \square) \mapsto (q_0, \square, \rightarrow)$: if we moved left and see a square, then we already riched the beginning of the word, and move one position right to pick the next letter.

Let us check that no cases are missing. In total, there are 6 states and 3 possible symbols. This should result in 18 transitions.

1.5. **Problem 5.** *Describe the Turing machine which recognises the following language:*

$$\{a^n b^n \mid n \geqslant 0\}$$

Idea. The leftmost symbol should be always a and the rightmost symbol should be always b. After verifying this condition we erase the leftmost and the rightmost bit and repeat the procedure. The word is accepted if after several iterations we obtain an empty word. Otherwise, if during some step the leftmost bit is not equal to a, or the rightmost bit is not equal to b, we reject.

1.6. **Problem 6.** *Describe the Turing machine which recognises the following language:*

$$\{a^{2^n} \mid n \geqslant 0\}$$

Idea. During the first step, we verify that the length of the word is divisible by 2 (the length is *even*). This can be done with 2 alternating states, and an additional state to return to the beginning.

Next, we want to remove exactly half of the word. This is done by putting the *markers* on the letters, by adding new symbols to the alphabet.

We put a marker $\circ$ on the left letter, and a marker $\star$ on the right letter. Then, we iteratively move the left marker one position right, and the right marker one position left until they meet. Then, we erase all the letters starting from the letter having the marker $\star$ on the right, and remove the marker $\circ$.

After doing this, the length reduced by a factor 2. This procedure is repeated until the length becomes odd (not divisible by 2). If the final length is 1, then the word is accepted, otherwise it is rejected.

1.7. **Problem 7.** *Describe the Turing machine which recognises the following language:*

$$\{ww \mid w \in \{0,1\}^*\}$$

Idea. Using the concept of markers, we can solve this problem.

- *Step 0.* Verify that the length of the input is divisible by 2.
- *Step 1.* Put a marker $\circ$ on the leftmost letter and the marker $\star$ on the rightmost.

- *Step 2.* Move the marker $\circ$ one position right, and the marker $\star$ one position left, until they meet.
- *Step 3.* Move the marker $\circ$ to the beginning of the word.
  At this stage, the markers $\circ$ and $\star$ mark the beginning and the middle of the input, respectively.
- *Step 4.* Verify that the symbols under the markers $\circ$ and $\star$ are equal. If they are not equal, finish in the rejection state. If they are equal, move each marker one position right.
- *Step 5.* If the marker $\star$ reached the end of the word, finish in the acception state.

1.8. **Remark.** The language from problem 5 still belongs to the set of *context-free* languages, while the languages from problem 6 and 7 do not belong there (can you prove this?)

2. TD 3

Definition 2.1. A problem belongs to the class NP (nondeterministic polynomial time) if it is a decision problem which can be solved in polynomial time on a non-deterministic Turing machine.

Alternatively, a problem belongs to the class NP if it has a proof of length polynomial on the size of the input which can be verified in polynomial time.

Definition 2.2. A problem X is polynomial-time Turing-reducible to a problem Y, if, given a subroutine that solves Y, one can construct a Turing machine M that uses this subroutine polynomially many times to solve X.

Remark 2.3. In our exercises, the number of calls of the subroutine is typically equal to 1.

Definition 2.4. A problem H is *NP-hard* if every problem $L \in \mathrm{NP}$ can be reduced in polynomial time to H.

Definition 2.5. A problem H is *NP-complete* if it belongs to NP and is NP-hard.

2.1. **Problem 1.** A *clique* in the non-oriented graph G is a subset of vertices that form a complete graph (in complete graph, there is an edge between each pair of vertices). The problem CLIQUE is to verify, given a graph G and a natural number k, if there exists a clique with k vertices in the graph G.

Show that 3-SAT $\leqslant_P$ CLIQUE.

Solution. We know that the instances of the problem 3-SAT are represented by 3-CNF. Consider an example of such 3-CNF:

$$(x \vee y \vee z) \wedge (\bar{x} \vee y \vee z)$$

The goal is to use the subroutine for the CLIQUE problem to be able to solve the 3-SAT problem.

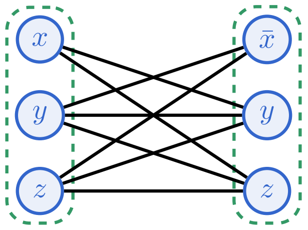

FIGURE 1. Formula $(x \vee y \vee z) \wedge (\bar{x} \vee y \vee z)$ represented by a graph

Let us construct a graph from the 3-CNF in the following way: each clause with 3 variables gives rise to 3 vertices, joined in a block; each vertex carries the label with the variable that was in the initial clause; every two vertices of this graph are connected unless the two vertices belong to the same block or the two vertices represent the negations of each other (see Figure 1).

In the scope of the reduction, given a formula with n variables and m clauses, we ask, whether there exists a clique of size m.

Proposition 2.6. If there is a clique of size m then the formula is satisfiable.

Proof. If there is a clique of several vertices, then from each block only one node will be chosen, because inside the block there are no edges. Since the clique has size m, then from each block exactly one vertex is chosen.

Moreover, each selected vertex corresponds to assigning value to the variable. It is not possible to assign simultaneously two values to the variable because there is never an edge $x\bar{x}$. Every pair of vertices belonging to the clique, represents a non-contradicting pair of assignments. Therefore, it is a valid assignment.

If for some literals no value was chosen, it can be chosen arbitrarily. □

Proposition 2.7. If the formula is satisfiable then there is a clique of size m.

Proof. If the formula is satisfiable then from every clause we can choose one variable whose value is true. From each clause we select exactly one variable, even if there is more than one possibility to choose. Note that between every two selected nodes there is an edge in the constructed graph because every variable is assigned some value, and therefore there is no edge $x\bar{x}$.

Consequently, this choice yields a clique of size m. □

Combining these two propositions, we obtain that the formula is satisfiable if and only if the constructed graph has a clique of size m.

2.2. **Problem 2.** A vertex-cover in a non-oriented graph G is a subset of vertices S' such that all the edges of the graph have at least one endpoint in the chosen subset S'. The problem VERTEX-COVER is to know, given a graph G and a natural number k, whether there exists a vertex-cover of size k in the graph G.

Show that CLIQUE $\leqslant_P$ VERTEX-COVER.

Solution. Consider a graph G. Let us construct a so-called *complementary graph* $\overline{G}$ in the following manner: there is an edge uv in the graph G between two vertices u and v, if and only if there is **no** edge between the vertices u and v in the graph $\overline{G}$.

Let $G = (V, E)$. Note that $S \subset V$ is a clique in the graph G if and only if $V \backslash S$ is a vertex cover in the graph $\overline{G}$.

2.3. **Problem 3.** An independent set in a non-oriented graph G is a subset of vertices S' such that there is no edge between any of the two vertices in S'. The problem INDEPENDENT-SET is to decide, given a graph G and a natural number k, whether there exist an independent set of k vertices in the graph G.

Show that CLIQUE $\leqslant_P$ INDEPENDENT-SET.

Solution. Using the same construction of the complementary graph $\overline{G}$ from the previous problem, we note that S is a clique in the graph G if and only if S is an independent set in the graph $\overline{G}$.

2.4. **Problem 4.** Consider a finite subset $E \subset \mathbb{N}$ and a number $c \in \mathbb{N}$. The problem SUBSET-SUM is to decide, given a set E and a number c, if there exists a subset $E' \subset E$ such that

$$\sum_{x \in E'} x = c.$$

Example: given $E = \{1, 3, 4, 7, 8\}$ and $c = 14$, we can choose $E' = \{3, 4, 7\}$ with $3+4+7 = 14$.

Show that VERTEX-COVER $\leqslant_P$ SUBSET-SUM.

Solution. Suppose that in the VERTEX-COVER problem in the graph $G = (V, E)$ there are vertices $v_1, v_2, \dots, v_n$ and the edges $e_1, e_2, \dots, e_m$.

For each vertex v_i we create a number x_i and for each edge e_j we create a number y_j in the following way:

- 1-st digit of x_i is equal to 1.
- $(j+1)$-st digit of x_i is equal to 1 if the edge e_j has an endpoint at vertex v_i, otherwise the digit is equal to 0.

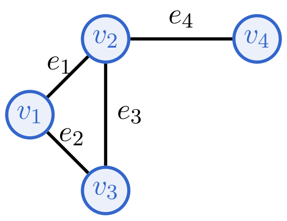

FIGURE 2. Example of graph for the VERTEX-COVER problem

- 1-st digit of y_j is equal to 0.
- $(j+1)$-st digit of y_j is equal to 1, all other digits are equal to 0.
- All the numbers are written in base $(n+m)$ so that the digits do not carry during the summation;
- The target sum c has first digit equal to the target size of the vertex cover subset, and all other digits are equal to 2.

For example, consider the graph from Figure 2. The numbers that we construct, can be found in the following table.

	v	e_1	e_2	e_3	e_4
x_1	1	1	1	0	0
x_2	1	1	0	1	1
x_3	1	0	1	1	0
x_4	1	0	0	0	1
y_1	0	1	0	0	0
y_2	0	0	1	0	0
y_3	0	0	0	1	0
y_4	0	0	0	0	1
s	k	2	2	2	2

Let us see what happens if we search for the vertex cover consisting of two vertices. We can take vertices v_3 and v_2 as an example.

The sum of the corresponding numbers x_2 and x_3 equals

$$x_2 + x_3 = \overline{21121}$$

In order to get the sum s it lacks additional 1's in positions $2, 3, 5$. This is achieved by adding the numbers e_1, e_2, e_4 to the sum.

In general, in order to acheve the sum s, we should always choose exactly k numbers from the set $\{x_1, \ldots, x_n\}$, and we also need to cover each digit of k starting from the second one, exactly twice. If, for a given edge e_j, only one of the endpoints belongs to the vertex cover, we need to additionally choose the number y_j, otherwise we don't choose this number.

It can be easily seen that the SUBSET-SUM in the given form has a solution if and only if the VERTEX-COVER problem has a solution.

3. TD 4

3.1. **Problem 1.** *Show that 3-SAT $\leqslant_P$ 4-SAT.*

Solution. We need to use our ability to solve 4-SAT problems to solve 3-SAT problems. Consider a 3-SAT problem. For each clause, choose one variable and add it one more time to the clause. The resulting problem now becomes 4-SAT, but it is equivalent to the initial 3-SAT.

Example:

$$F_1 = (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee w)$$

$$F_2 = (x \vee y \vee \bar{z} \vee \bar{z}) \wedge (x \vee \bar{y} \vee w \vee w)$$

3.2. **Problem 2.** *Show that PARTITION $\leqslant_P$ SUBSET-SUM.*

Solution 1. Since PARTITION $\in$ NP and it was shown that SUBSET-SUM is NP-complete, there exists a reduction from PARTITION to SUBSET-SUM.

Solution 2. Consider an instance of PARTITION problem. This is a set E consisting of non-negative integers, and the goal is to split E into the two subsets A and $B = E\backslash A$ such that the sum of the elements in A is equal to the sum of the elements in B.

Example: $E = \{1, 2, 3, 4, 5, 6, 7\}$. This set can be separated into $A = \{3, 4, 7\}$ and $B = \{1, 2, 5, 6\}$. The sum of the numbers in each set is equal to

$$3 + 4 + 7 = 1 + 2 + 5 + 6 = 14.$$

Let M be equal to the sum of all the numbers in E. We need to choose a subset of numbers whose sum is equal to $M/2$. If M is not divisible by 2, then there is no solution. Otherwise we solve the problem SUBSET-SUM with the initial set E and the target sum $M/2$.

3.3. **Problem 3.** *Show that CYCLE-HAM $\leqslant_P$ TRAVELLING-SALESMAN.*

Solution. The problem of finding a Hamiltonian cycle is to find a cycle which contains all the vertices of a given graph and passes through each vertex exactly once. The problem TRAVELLING-SALESMAN is, given a graph G with non-negative weights on the edges, and a number M, find a cycle with sum of the weights at least M.

In order to reduce CYCLE-HAM to TRAVELLING-SALESMAN, we equip the initial graph with weights 1 for each of the edges, and set the target cost equal n, where n is the number of vertices.

It is easy to see that in the weighted graph, there exists a cycle with total weight at least n if and only if the initial graph contains a Hamiltonian cycle.

3.4. **Problem 4.** *Show that 3-COLOR $\leqslant_P$ 4-COLOR.*

See problem 6 from the mid-term exam 2017.

3.5. **Problem 5.** *Show that SUBSET-SUM $\leqslant_P$ PARTITION.*

Solution. Consider the instance of the problem SUBSET-SUM which is a set of non-negative integers E and the target non-negative integer S. Let M be the sum of all the numbers from E.

If $M > S$, then there is no solution. If $M = S$, then the solution trivially exists.

Let us construct a new set $E' = E \cup \{M + S, 2M - S\}$. It can be shown that the set E' can be partitioned into two subsets with equal sum if and only if there is a subset of elements of E summing to S. The idea is to show that if E' can be partitioned into two parts, then the numbers $M + S$ and $2M - S$ should belong to two different subsets, otherwise the sum in this part containing both $M + S$ and $2M - S$ will be too large.

If these two numbers belong to the different parts, then the numbers from the part containing $2M - S$ sum to S.

4. Mid-term exam 2017

4.1. **Problem 1.** In the frame of the proof that SAT $\leqslant_P$ 3-SAT, give an instance of the 3-SAT formula which is equivalent in terms of satisfiability to the SAT formula

$$(x \to y) \wedge (\bar{x} \leftrightarrow y)$$

Solution. After transforming the SAT formula to the CNF we obtain

$$(\bar{x} \vee y) \wedge (x \vee y) \wedge (\bar{x} \vee \bar{y}).$$

This can be transformed into 3-CNF by doubling the literals in each clause:

$$(\bar{x} \vee y \vee y) \wedge (x \vee y \vee y) \wedge (\bar{x} \vee \bar{y} \vee \bar{y}).$$

4.2. **Problem 2.** In the frame of the proof that 3-SAT $\leqslant_P$ 3-COLOR, describe a graph in which the problem of 3-colorability enables to solve the satisfiability of

$$(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z})$$

Solution. In the reduction procedure, we introduce vertices with labels $x, \bar{x}, y\bar{y}, z, \bar{z}$, and also three special vertices T, F, N meaning, respectively, True, False and Neutral.

The vertices T, F, N are connected into the triangle, and for each variable x, the vertices $x, \bar{x}, N$ are also connected by a triangle.

Then, for each clause $(a \vee b \vee c)$ an OR-gadget is created. Consult your lectures for the explicit form of the OR-gadget because there are several possible ways to construct it.

4.3. **Problem 3.** *Describe (picture and explanation) a Turing machine that decides whether a word from* $\{0, 1\}^*$ *has exactly 3 zeros.*

Idea. For precise descriptions of the solutions of similar problems, see section TD2. For this particular problem, we need the states q_0, q_1, q_2, q_3 which represent the number of zeros that are currently found. The head of the Turing machine runs from left to right, switching from the state q_i to q_{i+1} only if it meets symbol 0.

- If the machine is in the state q_3 and meets symbol 0, then it finishes in the rejection state.
- If it reaches the end of the word in the state q_3 then the word should be accepted.
- If it reaches the end of the word in a state other than q_3 then the word should be rejected.

4.4. **Problem 4.** *Describe a Turing machine that takes a word in* $\{0, 1\}^*$ *and returns a word having a cyclic shift, finishing in the state of acception.*

Example: $\star$1000101 becomes $\star$0001011. The symbol 1 on the right moves to the left. It is forbidden to replace the symbol $\star$.

Idea.

- The Turing machine should go to the end of the word, remember the last digit (i.e. turn into one of the dedicated states r_0 or r_1).
- Then the last digit is erased.
- Iteratively, the head of the Turing machine moves one symbol left, inserting the remembered symbol and remembering the erased symbol.
- When the head reaches $\star$, it turns into one of the two specially dedicated states p_0, p_1, which skips until of the word carrying, respectively, symbol 0 or 1 in the memory.
- When the head meets an empty symbol, it inserts the remembered symbol and finishes in the acception state.

4.5. **Problem 5.** *Describe a Turing machine that takes a word in* $\{0, 1\}^*$ *and replaces all* 0 *by* 10*, and each* 1 *by* 01.

Example: $\star$10 becomes $\star$0110.

Idea.

- Put a marker on the first symbol of the input, and after the last symbol of the input.
- Iteratively move the markers by one position right until the first marker reaches the end of the word.
- Now the second marker marks twice the length of the input.
- Erase the last letter of the input and remember it. Go until the marker.
- Replace the marker with 10 in case of zero, or by 01 in case of one.
- Return back to the end of the input (one letter at the end is already erased).
- Repeat the procedure, iteratively erasing one symbol at the end of the input, skipping the blank symbols and inserting the transformed symbols.
- Finish when the length of the input becomes zero, i.e. when the symbol $\star$ is reached.

4.6. **Problem 6.** *Show that 4-COLOR $\leqslant_P$ 5-COLOR.*

Solution. Consider an instance of the problem 4-COLOR. This is a graph $G = (V, E)$. Let us construct another graph $G' = (V', E')$ with additional vertex

$$V' = V \cup \{x\}$$

and connect all the vertices with this new additional vertex x:

$$E' = E \cup \bigcup_{v \in V} \{(v, x)\}$$

where (v, x) represents the edge between the vertices v and x.

It is straightforward to show that the graph G' is 5-colorable if and only if the graph G is 4-colorable.

4.7. **Problem 7.** *Show that 3-COLOR $\leqslant_P$ SAT.*

Solution 1. Let us take an instance of 3-COLOR problem which is a graph $G = (V, E)$. For each vertex v_i we create three variables x_i, y_i, z_i. Assigning value TRUE to some of the variables means that vertex v_i is colored in one of the three corresponding colors.

For example, if x_i is TRUE, then the vertex v_i is colored in the first color, if y_i is TRUE, then it is colored in the second color, and finally, if z_i is TRUE, it is colored in the third color.

For each vertex v_i we need to create a logical condition that exactly one of the variables x_i, y_i, z_i is TRUE:

$$x_i \vee y_i \vee z_i = 1, \quad x_i \wedge y_i = 0, \quad y_i \wedge z_i = 0, \quad z_i \wedge x_i = 0.$$

The main condition of the 3-colorability says that along each edge connecting vertices u and v, so for each edge $e_k = (v_i, v_j)$ we add a set of conditions saying that v_i and v_j cannot be colored the same color:

$$x_i \wedge x_j = 0, \quad y_i \wedge y_j = 0, \quad z_i \wedge z_j = 0.$$

It is straightforward to see that the graph G is 3-colorable if and only if the constructed formula is satisfiable.

Solution 2. Since 3-COLOR $\in \mathcal{NP}$ and SAT is $\mathcal{NP}$-complete (as was demonstrated during the course), there exists a reduction from 3-COLOR to SAT.