Languages, Automata and Computations 2

Classes 1

We will consider ω-regular languages (shortly: regular) with Büchi and Muller acceptance condition.

Exercise 1.
Are the following languages regular:

1. prefix of v belongs infinitely often to the fixed regular language of finite words $L \subseteq \Sigma^*$;
2. word v contains infinitely many infixes of the form ab^pa, where p is prime;
3. word v contains infinitely many infixes of the form ab^pa, where p is even;
4. word v contains arbitrary long infixes in the fixed regular language of finite words L;
5. prefix of v belongs infinitely often to the fixed language of finite words $L \subseteq \Sigma^*$ (not necessarily regular).

Solution
Solutions of the points:

1. YES. Let $\mathcal{A}$ be an automaton for L. We make the same automaton with the same final states and Büchi acceptance condition.
2. NO. Assume that yes and $\mathcal{A}$ is an automaton for this language. Let $\mathcal{A}$ has n states and let $p > n$ be some prime number. The word $(b^pa)^\omega$ belongs to L, so $\mathcal{A}$ has an accepting run on it. Note that in every block b^p some two states are the same. We pump the part b^k between them p times, so that the block has now the length b^{p+kp}, which is not prime. The now word is accepted by $\mathcal{A}$, because it has an accepting run (which came out from the pumping of an old run), but no block has prime length.
3. YES. It suffices to count length of the block b^k modulo 2 and go into an accepting state on a which finishes such a block.
4. NO. Consider $L = ab^*a$. This language contains no ultimately periodic word, so it would have to be empty to be regular.
5. NO. Let us fix some infinite word u, which is not ultimately periodic. Let L be all its prefixes. When prefix of v belongs infinitely often to L if and only if $v = u$. However language $\{u\}$ is not regular. By the way note that the language $\{w\}$ is regular iff w is ultimately periodic.

Exercise 2.
Show that language of words "there exists a letter b" cannot be accepted by

a nondeterministic automaton with Büchi acceptance condition, where all the states are accepting (but possibly transitions over some letters missing in some states).

Solution
By reading a^k for any $k \in \mathbb{N}$ we cannot get blocked. Therefore word a^ω also cannot get blocked, which means that it is accepted by the considered automaton, contradiction.

Exercise 3.
Show that language "finitely many occurrences of letter a" cannot be accepted by a deterministic automaton with Büchi acceptance condition.

Solution
Assume towards a contradiction that it is accepted by some automaton with n states. Let $w = (ab^n)^\omega$. For any prefix of it of the form $(ab^n)^k$ there should be an accepting state among the last $n+1$ states. Indeed, otherwise a word $(ab^n)^k b^\omega$ would not be accepted. Therefore a run over w visits infinitely many times an accepting state, which means that w is accepted. On the other hand it does not belong to the language, as it has infinitely many letters a. Contradiction.

Exercise 4.
Show that every language accepted by some nondeterministic automaton with Muller acceptance condition is also accepted by some nondeterministic automaton with Büchi acceptance condition.

Solution
We have an automaton $\mathcal{A}$ with Muller condition, we will be trying to make an automaton $\mathcal{A}'$ with Büchi condition such that $L(\mathcal{A}') = L(\mathcal{A})$. For every $S \in \mathcal{F}$ we will do a separate gadget in automaton $\mathcal{A}'$ such that acceptance in this gadget is if and only if $\inf(\rho) = S$. At the beginning we just make a copy of $\mathcal{A}$, but such that no states are accepting. Beside that for every $S \in \mathcal{F}$ we add a gadget $\mathcal{A}_S$. The idea is that automaton $\mathcal{A}'$ will jump into the gadget $\mathcal{A}_S$ if it wants to choose that $\inf(\rho) = S$ and now is exactly this moment from which on only states from S will occur. Let $S = \{q_1, \ldots, q_k\}$.

Observe now that if $\inf(\rho) = S$ and there are no states outside of S in ρ then states occurring in ρ have also an infinite subsequence of the form $(q_1 q_2 \cdots q_k)^\omega$. Thus we can just investigate whether there exist such a subsequence. Gadget $A_§$ will be the following. It contains $|S|$ copies of $\mathcal{A}$: $\mathcal{A}_{S,1}, \ldots, \mathcal{A}_{S,|S|}$. The copy $\mathcal{A}_{S,i}$ has only one accepting state: q_i. In the copy $\mathcal{A}_{S,i}$ transitions are like in $\mathcal{A}$ with the only exception that from state q_i we go to the next copy: $\mathcal{A}_{S,(i+1) \mod |S|}$. Now we can easily observe that ρ visits infinitely many times accepting state iff it infinitely many times changes a copy. Therefore it has a subsequence of states of the form $(q_1 q_2 \cdots q_k)^\omega$, so indeed $\inf(\rho) = S$.

Exercise 5.
Assume that we have changed the acceptance condition into such which investigates which sets of transitions are visited infinitely often. Does it affect the expressivity of automata? How it is for Büchi acceptance condition? And how for Muller acceptance condition?

Solution

In both cases (Büchi and Muller) expressivity does not change. Therefore we have to prove four facts: (1) condition with states can be implemented on transitions, (2) condition with transitions can be implemented on states, both points for both Büchi and Muller acceptance conditions.

Let us start

1. We first implement states on transitions for Büchi condition. It is very easy, simply these transitions are final which finish into previously final states.

2. Now we implement transitions on states for Büchi condition. Let language L be accepted by automaton $\mathcal{A}$ with Büchi acceptance condition on transitions. We make an automaton $\mathcal{A}'$, which has two copies of every state in $\mathcal{A}$. To one copy go all the accepting transitions, while to another one go all the non accepting ones. The outgoing transitions are identical in both copies, the same as in $\mathcal{A}$. All the copies with accepting incoming transitions are final, while the other not (some of the final states may not be reachable, but this is not the problem).

3. Now we implement states on transitions for Muller acceptance condition. This is also easy, set of transitions is accepting iff the set of states into which they go is accepting.

4. Now we implement transitions on states for Muller acceptance conditions. Let automaton $\mathcal{A}$ with Muller condition on transitions accept $L = L(\mathcal{A})$. We make $\mathcal{A}'$ as follows. Every state of $\mathcal{A}$ is split into as many copies in $\mathcal{A}'$ as it has incoming transitions in $\mathcal{A}$. The state of $\mathcal{A}'$ is accepting if the incoming transition in $\mathcal{A}$ was accepting.

Exercise 6.
Show that nonemptiness is decidable for automata with Muller acceptance condition.

Solution
It is enough to check whether for some $S \in \mathcal{F}$ there exist a run ρ of an automaton in which $\inf(\rho) = S$, where $\inf(\rho)$ is the set of states which occur infinitely often in ρ. We do this separately for every $S \in \mathcal{F}$. We check whether we graph with only states from S is strongly connected and whether some state from S is reachable from some initial state (now in the situation where we have all the states).

Classes 2

Today we continue infinite words. We do a bit of connection to topology and other stuff.

Exercise 7.
Let us define a metric on infinite words: $d(u, v) = \frac{1}{2^{\mathrm{diff}(u,v)}}$, where $\mathrm{diff}(u, v)$ is the smallest index on which u and v differ. Language L is open (in this metric) if for every $w \in L$ there exist some open ball centered in w which is included in L (so the standard definition). Prove that the following conditions are equivalent:

1. language L is open;
2. language L is of the form $K\Sigma^\omega$ for some $K \subseteq \Sigma^*$.

Solution
We will show equivalence of (1) and (2), so two implications.

First (1) $\Rightarrow$ (2). If L is open then around every $w \in L$ there is a ball L_w with positive radius such that $L_w \subseteq L$. Therefore $L = \bigcup_{w \in L} L_w$. Observe however that $L_w = f(w)\Sigma^\omega$, where $f(w)$ is some finite prefix of w.Thus $L = \bigcup_{w \in L} f(w)\Sigma^\omega$, so $L = K\Sigma^\omega$ for $K = \bigcup_{w \in L} f(w)$.

Now (2) $\Rightarrow$ (1). Let $w \in L = K\Sigma^\omega$. Therefore $w = uv$, where $u \in K$. Thus for any $v' \in \Sigma^\omega$ we have $uv' \in L$. Therefore the ball centered in w and radius equal $\frac{1}{2^{|u|+1}}$ is included in L.

Exercise 8.
Let us define a metric on infinite words: $d(u, v) = \frac{1}{2^{\text{diff}(u,v)}}$, where diff$(u, v)$ is the smallest index on which u and v differ. Language L is open (in this metric) if for every $w \in L$ there exist some open ball centered in w which is included in L (so the standard definition). Prove that the following conditions are equivalent for a regular language L:

1. language L is open;
2. language L is of the form $K\Sigma^\omega$ for some $K \subseteq \Sigma^*$;
3. language L is of the form $K\Sigma^\omega$ for some regular $K \subseteq \Sigma^*$.

Solution
We will first show equivalence of (1) and (2), so two implications.

First (1) $\Rightarrow$ (2). If L is open then around every $w \in L$ there is a ball L_w with positive radius such that $L_w \subseteq L$. Therefore $L = \bigcup_{w \in L} L_w$. Observe however that $L_w = f(w)\Sigma^\omega$, where $f(w)$ is some finite prefix of w.Thus $L = \bigcup_{w \in L} f(w)\Sigma^\omega$, czyli $L = K\Sigma^\omega$ for $K = \bigcup_{w \in L} f(w)$.

Now (2) $\Rightarrow$ (1). Let $w \in L = K\Sigma^\omega$. Therefore $w = uv$, where $u \in K$. Thus for any $v' \in \Sigma^\omega$ we have $uv' \in L$. Therefore the ball centered in w and radius equal $\frac{1}{2^{|u|+1}}$ is included in L.

Now we will show equivalence of (1) and (3). Implication (3) $\Rightarrow$ (1) works the same as (2) $\Rightarrow$ (1), so we will focus on implication (1) $\Rightarrow$ (3).

Let us consider an automaton $\mathcal{A}_L$ of language L, where $\mathcal{A}_L$ is a deterministic automaton with Muller acceptance condition. Let $Q_U \subseteq Q$ be a subset of states such that $q \in Q_U$ iff $L(q) = \Sigma^\omega$. Let $L' \subseteq \Sigma^*$ be the set of finite words accepted by automaton $\mathcal{A}_L$, where final states are Q_U. We will show that $L = L'\Sigma^\omega$. We have $L \supseteq L'\Sigma^\omega$, because it is easy to find a run in $\mathcal{A}_L$ for every word from $L'\Sigma^\omega$. We simply first go like in automaton for L' and then arbitrarily and this is an accepting run for $\mathcal{A}_L$. On the other hand let us take $w \in L$ and its run. After some its prefix we will already be in the ball L_w, which means that not depending on the suffix everything is accepted, so we are in the state from Q_U. This gives us a division into prefix from L' and the rest.

Exercise 9.
Consider a transducer, which defines a function $f : \Sigma^\omega \to \Gamma^\omega$ and metrics on Σ^ω and Γ^ω defined as $d(u, v) = \frac{1}{2^{\text{diff}(u,v)}}$. Show that such an f is continuous.

Solution
It is easy to do this from definition. Let us take some two words, which are close to each other, so they agree on some prefix, say of length n. Then their images will also agree on the prefix of length n. Therefore by definition: if we want that images agree on prefix of length n it is enough to take arguments which agree on prefix of length n.

Exercise 10.
Let UP be the set of ultimately periodic words, i.e. $UP = \{uv^\omega \mid u, v \in \Sigma^*\}$. Show that for all regular languages K, L of infinite words if $K \cap UP = L \cap UP$ then $K = L$.

Solution
Consider $K \setminus L$. We have that $(K \setminus L) \cap UP = \emptyset$ and $K \setminus L$ is regular. Therefore by the fact shown before (at some previous exercises) we know that $K \setminus L = \emptyset$. Similarly we show that $L \setminus K = \emptyset$, which together implies that $K = L$.

Exercise 11.
We will look for a candidate for Myhill-Nerode relation for infinite words, i.e. an equivalence relation $\sim_L$ such that $\sim_L$ has finite index iff L is regular. Check whether this fact is true for the following relations

1. $\sim_L \subseteq \Sigma^* \times \Sigma^*$ such that $u \sim_L v$ iff for all $w \in \Sigma^\omega$ it holds $uw \in L \iff vw \in L$;
2. $\sim_L \subseteq \Sigma^\omega \times \Sigma^\omega$ such that $u \sim_L v$ iff for all $w \in \Sigma^*$ it holds $wu \in L \iff wv \in L$;
3. $\sim_L \subseteq \Sigma^* \times \Sigma^*$ such that $u \sim_L v$ iff for all $w \in \Sigma^\omega$ and $s, t \in \Sigma^*$ it holds $uw \in L \iff vw \in L$ and $s(ut)^\omega \in L \iff s(vt)^\omega \in L$.

Solution
We will show that none of these relations has finite index iff L is regular. In all cases if L is regular then $\sim_L$ has a finite index. We will first show this. Consider an automaton $\mathcal{A}$ with Büchi condition for L. In $\sim_L$ we just have to remember

1. which states of $\mathcal{A}$ one can reach via a word u;
2. from which states of $\mathcal{A}$ there exists an accepting run via a word u;
3. as in 1. and in 2. and additionally for which pair of states $p, q \in Q(\mathcal{A})$ there exists a run via a word u which goes from p to q and a) has an accepting state b) has no accepting state.

Now we show that there exists nonregular languages L such that $\sim_L$ has finite index.

1. every prefix independent language, for example: language from exercises 1.2 (u contains infinitely many infixes of the form ab^pa, where p is prime). For such a language $\sim_L$ has only one equivalence class;
2. also every prefix independent language, as language from 1.2 works. For such a language relation $\sim_L$ has two classes ($wu \in L$ or $wu \notin L$, this does not depend on w);

3. language similar like from exercise 1.4 works (u contains arbitrary long infixes of the form ba^*b). It is not regular. Definitely $uw \in L$ does not depend in u. It is enough to know whether u contains any b (if yes then $s(vt)^\omega \notin L$) and additionally if not whether u is empty (it matters for empty t).

In general there exists no reasonable relation with this property.

Classes 3

Today we will work on games.

Exercise 12.
We will now show an example of a game, which is not determined. We will construct this example by a sequence of a few exercises. First consider a following riddle. There are infinitely many players (countable many), every one has a hut: either white or black. Everybody sees the color of everybody else hut, but not of his own. Everybody should say what is the color of his hut, such that only finitely many players will make a mistake. They can fix a strategy before, but they cannot tell anything to each other after they will see the huts. What is the winning strategy?

Solution
Hint: consider the following relation on infinite 0-1 sequences $w \sim w'$ iff they differ on finitely many places. We can treat a hut configuration as an infinite 0-1 sequence. Relation $\sim$ is an equivalence relation. A strategy is to fix one element in every equivalence class. Everybody can easily recognize what is the equivalence class. Then everybody says the corresponding number of this one distinguished element in the equivalence class. Only finitely many players will make a mistake.

Exercise 13.
Define a function $\text{xor} : \{0,1\}^\omega \to \{0,1\}$, called an *infinite xor*, such that changing one bit or an argument changes the result.

Solution
In every equivalence class C of $\sim$ choose one element v_C and put $\text{xor}(v_C) = 0$. For every $v \in C$ put $\text{xor}(v) = \text{fin-xor}(v \otimes v_C)$, where $\otimes$ is the standard bit xor and fin-xor outputs 0 iff an argument has even number of 1 (argument of fin-xor has to have a finite number of ones). It is easy to verify that xor is indeed an infinite xor.

Here I presented the strategy stealing argument, this year by considering a riddle about the chocolate game (rectangular board, players eating parts of chocolate, the one eating last part loses).

Exercise 14.
Define a non determined game.

Solution
Hint 1: use an infinite xor. Hint 2: the game is as follows. There are two players, they construct an infinite 0-1 sequence. Player One wins if xor of constructed

element is 1, otherwise Zero wins. Zero and One construct this infinite sequence w by delivering in an alternating manner a finite 0-1 sequences and appending it to the currently constructed prefix of w. Assume that Zero starts.

Observe now that this game is not determined. We will show that no player has a winning strategy. Assume first that One has a winning strategy σ. We will show how Zero can sometimes win against this strategy. Consider two plays P_1 and P_2. Let Zero play 0 in play P_1 and the response of One in P_1 be v_0. Then Zero plays $1v_0$ in P_2 and response of One in P_2 is some v_1. Then Zero plays v_1 in P_1 and the response of One in P_1 is some v_2. Then Zero plays v_2 in P_2 and the response of One in P_2 is some v_3. In that way Zero copies responses of One to another play. Then in P_1 the constructed play will be of the form $0v_1v_2v_3\cdots \in \{0,1\}^\omega$ and in P_2 it will be of the form $1v_1v_2v_3\cdots \in \{0,1\}^\omega$. So the different player will win these plays, contradiction with the assumption that σ is a winning strategy for One.

Similarly we can get a contradiction with the assumption that Zero has a winning strategy. Zero starts with some v both in P_1 and in P_2. Then One plays 0 in P_1, the response of Zero is v_1 and One plays $1v_1$ in P_2. Then he copies Zero's responses as before and the results of these plays will be different in P_1 and P_2. This leads to the contradiction.

So indeed this xor-game is not determined.

Classes 4

Mikołaj was making exercises about parity games.

Exercise 15.
Show that solving one player parity game is in PTIME.

Solution
This amounts to checking whether there is a reachable cycle with the largest rank of the given parity.

Exercise 16.
Show that deciding which player has a winning strategy from a given vertex in parity game is in $\mathrm{NP} \cap \mathrm{coNP}$.

Solution
We will use the fact that in parity games winning strategies can be chosen to be positional. To show presence in NP we do the following. For a vertex in which player i wins we guess a positional strategy for player i (an edge from every i-vertex, so a polynomial object). To check that it is indeed winning for i we have to choose that in a graph which appeared after fixing a strategy for i the player i indeed wins. It is enough to check that there is no reachable loop in which the smallest rank has parity $1-i$, which can be easily done in polynomial time.

In order to show presence in coNP we have to show that the complement is in NP. It is easy, as the complement is existence of winning strategy for player $1-i$ and since the game is almost symmetric this is also in NP.

Exercise 17.
Consider the following finite game on finite graph V together with function

rank : $V \to \mathbb{N}$. Player play as usual till the moment when some vertex repeats on the play, so till the first loop is created. Then we look at the smallest rank on this loop and the player with the corresponding parity wins. Prove that player i in the presented game wins iff player i wins in the parity game on the same arena. In other words: this is a finite version of the parity game.

Solution
Assume that i wins in the parity game. Then he has a winning strategy which is positional. Let i play the same strategy in the loop-game. Consider the first closed loop. If the smallest rank on this loop would have parity $1-i$ it would mean that there exists a strategy of $1-i$ in the parity game, which forces the play to go through that loop all the time and the result would be that player i would lose. So the smallest rank on the loop has the parity i, so i also has a winning strategy in the loop game. If $1-i$ wins in the parity game then in the same way we show that $1-i$ wins the loop-game. So indeed these two games are equivalent.

Classes 5

Today we start finite tree automata and Monadic Second Order logic.

Regular languages of trees.

We will consider here, for simplicity, very special kind of trees. These are trees such that every node has either two or zero children. Similar theory can be quite easily adapted to another kind of trees, but this would deliver some additional technicalities, which we prefer to avoid.

Exercise 18.
Consider the following four variants of tree automata: deterministic/nondeterministic top-down/bottom-up automata. States are written on edges and a transition is of the form (p, a, p_L, p_R), where state p is above the node, letter a is in the node and states p_L and p_R are on the transitions to left and right child, respectively. Which of these four variants are equivalent and which not?

Solution
It is convenient to think about the run of automaton a bit differently than before (in finite word case). Before we were usually thinking that automaton is processing a word from left to right and assigns to every edge (between letters) a state. Now it is better to think more declarative. Think that we label all the edges simultaneously and this labeling is correct if it is consistent with transition relation. Then we easily see that nondeterministic top-down and bottom-up automata has the some expressivity, as they actually have the same declarative definition.

We will now show that deterministic bottom-up variant is equivalently expressive, but deterministic top-down variant is weaker. For focus on deterministic bottom-up variant.

We say that automaton is bottom-up deterministic if for every $p_1, p_2 \in Q$ and $a \in A$ there exists at most one $p \in Q$ such that (p, a, p_1, p_2) is a transition.

We will show that for every nondeterministic automaton there exists an equivalent bottom-up deterministic automaton. We just apply a subset construction bottom-up. A new state will be the set of old states. An edge will be in the new state $S \subseteq Q$ if it can be in all old states $q \in S$ (there exists a labeling). One can easily see that this information can be updated bottom-up deterministically. A set of states is final iff it contains at least one old final state.

Now let us show that deterministic top-down variant is weaker. We will show that it cannot recognize the language: there exists an a-labeled node in the tree. This language can be easily recognize by a nondeterministic variant. Assume that there is some deterministic top-down automaton $\mathcal{A}$ recognizing this language with initial state $q_0 \in Q$. There is some transition (q_0, b, q_L, q_R). If there is an a in the left tree, but no a in the right tree $\mathcal{A}$ should reach final states everywhere, so there is an accepting run from q_R on the right subtree even if there is no a there. Similarly there is an accepting run from q_L on the left subtree even if there is no a there. So $\mathcal{A}$ can accept also trees such that there is no a anywhere in the tree.

We call a tree language *regular* if it can be accepted by some nondeterministic tree automaton or some bottom-up deterministic tree automaton.

Exercise 19.
Are the following tree languages regular:

1. trees with even number of nodes;
2. trees with even number of a-nodes;
3. trees over leaf alphabet $0, 1$ and internal alphabet $\vee, \wedge$ which evaluate to true as a boolean expression;
4. balanced trees (every path to the leaf has the same length).

Solution
In cases 1.-3. it is easy to show a nondeterministic automaton. Think that it goes bottom-up (it is usually a better perspective). In 1. it counts number of nodes modulo 2. Actually in 1. a tree which we consider is never accepted, because it always has an odd number of nodes. In 2. it counts number of a-nodes modulo 2. In 3. in remembers the boolean value of the subtree. In the case 4. language is not regular. It is easy to see. Consider the deterministic bottom-up automaton. Let q_k be a state assigned to a complete binary tree of depth k. Let our automaton have n states. Then by pigeonhole principle some two among the trees $q_1, \ldots, q_{n+1}$ have the same state, say q_i and q_j. Then tree $a(q_i, q_i)$ and tree $a(q_i, q_j)$ will behave the same with respect to this automaton, but they shouldn't: the first one is in the language, while the second one not.

Exercise 20.
Let a *yield* of a tree the the word composed from labels of its leafs read in the infix order. Show that for any $L \subseteq \Sigma^*$ the following are equivalent

1. L is context-free;
2. L is the set of yields of some regular tree language.

Solution
First implication from 1. to 2. Just consider a grammar in Chomsky normal form for L and the regular language of all its derivation trees. We can easily see that yield of a derivation is the derived word. So indeed the set of yields of the regular language of derivations is L.

Implication from 2. to 1. is also not much harder. Just build a context-free grammar in Chomsky normal form from our regular tree language. For every transition (p, a, q, r) make a rule $X_p \longrightarrow X_q X_r$ in the grammar and for every (p, a, q, r), where q and r are accepting make a rule $X_p \longrightarrow a$ in the grammar. Then the language of the grammar is exactly the set of yields of our regular tree language.

Monadic Second Order Logic.

In Second Order Logic (SO) one can quantify also over relations. In Monadic Second Order Logic (MSO) one can quantify over monadic relations, which are actually sets. In Existential Second Order Logic ($\exists$SO) one can write $\exists_{R_1,\ldots,R_n}\phi$, where R_i are any relations and ϕ is a first order sentence (which of course may use R_i).

Exercise 21.
Show that language of words of even length is expressible in MSO.

Solution
We will use sets S and T to mark odd and even positions, respectively. We will also use macros first(x) defined as $\forall_{y\in X} x \leq y$, last$(x)$ defined as $\forall_{y\in X} x \geq y$ and next(x, y) defines as $(x \leq y) \wedge (\forall_{z\in X} \neg(x < z \wedge z < y))$. The whole formula looks as follows

$$\exists_{S,T\subseteq X}(\forall_{x\in X} x \in S \vee x \in T) \wedge$$
$$(\forall_{x\in X} \neg(x \in S) \vee \neg(x \in T)) \wedge$$
$$(\forall_{x\in X}(\text{first}(x) \Rightarrow x \in S) \wedge (\text{last}(x) \Rightarrow x \in T)) \wedge$$
$$(\forall_{x,y\in X}(\text{next}(x, y) \Rightarrow (x \in S \iff y \in T))).$$

Exercise 22.
Show that it is expressible in MSO that a graph is connected.

Solution
The idea is that graph is connected if there is no partition of its vertices such that both parts are nonempty, but there is no edge between the part. We write the following formula:

$$\neg\exists_{S,T\subseteq V}$$
$$\forall_{x\in V}\big((x \in S \vee x \in T) \wedge (x \notin S \vee x \notin T)\big) \wedge$$
$$\forall_{x,y\in V}(x \in S \wedge y \in T) \Rightarrow \neg E(x, y)$$

Exercise 23.
Show that it is expressible in MSO that a graph is a forest.

Solution
The idea is that graph is a forest if there is no cycle. And there is a cycle if there is set of vertices such that every vertex from this set has exactly two neighbors from this set. We don't write a formula here, but it is not hard to write it.

Classes 6

Today we continue logic.

Exercise 24.
Show that for finite trees MSO(lson, rson) = Reg.

Solution
Similarly as for finite words, but there are some technical details (not causing any real trouble).

An automaton on infinite (binary) trees with parity condition works as follows. It has transitions of the form (p, a, q_ℓ, q_r) which means that if automaton is in state p and a node is labelled by a then it can send q_ℓ to the left child and q_r to the right child. The root should be labelled by some fixed initial state. Acceptance condition is as follows: on every path the parity condition should be satisfied.

Exercise 25.
Show that the following languages of infinite trees are regular (so accepted by some nondeterministic automaton):

1. on every path some regular language L;
2. there is somewhere letter a;
3. in every subtree there is letter a.

Solution
We construct automata as follows.

1. We take a deterministic parity automaton for L and on every path we use this automaton. Note that it is important that this automaton is deterministic, as it should behave the same on the prefix of two paths, which agree on some (finite) prefix.
2. This one is simple, we just nondeterministically guess where is the letter a. State q has to send into one child state q (still searching for a) and into one child state q' (accepting forever).
3. This one is harder. Let assume wlog. that $\Sigma = \{a, b\}$. The automaton is as follows. It has two states: accepting q_a and not accepting q_r. We have transitions (q, a, q_a, q_a) for $q \in \{q_a, q_r\}$ and transitions (q, b, q_a, q_r), (q, b, q_r, q_a) for $q \in \{q_a, q_r\}$. In other words if we see letter a we send accepting states into both child and otherwise only to one child. Clearly if there is a subtree in which there is no letter a then for every run (so labeling) in this subtree there is an infinite path without accepting state.

Indeed, we just always go down into the child, where the state q_r was sent. Now we show that for every tree such that in every subtree there is a letter a there exists an accepting run. We construct it. For every node let us choose some its descendant, which is labelled by a. Say for example that it is a shallowest descendant which is leftmost among the shallowest. Then if for a node u descendant v is chosen that for a node u', a child of u, which is an ancestor of v also descendant v is chosen. Then we construct a run: for every node the edge going down in the direction of chosen descendant is labelled by q_r and the other one is labelled by q_a. This is really an accepting run. On every path either we follow the path to chosen descendant and after a finite time we hit letter a and thus q_a or we deviate from the path to chosen descendant and then we immediately have state q_a. Thus on every path we always have a finite time till the state q_a, so all the paths are accepted. Clearly acceptance and nonacceptance of states q_a and q_r can be implemented on ranks.

Classes 7

Today we start tree-width.

Tree decomposition of a graph $G = (V, E)$ is a labelled tree t such that:

1. every node n of t is labeled by its *bag* $B_n \subseteq V$,
2. for every vertex $v \in V$ there exists a bag B_n such that $v \in B_n$,
3. for every edge $(u, v) \in E$ there exists a bag B_n such that $u, v \in B_n$,
4. for every vertex $v \in V$ set S of nodes of tree t defined as $S = \{n \mid v \in B_n\}$ is consistent in t.

Tree width of decomposition t is $\max_{n \in t} |B_n - 1|$. Minus one is because we want to have tree width of a tree to be equal one. Tree width of a graph G, denoted $\mathrm{tw}(G)$, is the minimal tree width of its decomposition.

Exercise 26.
Show that $\mathrm{tw}(G) = 1$ iff G is a forest.

Solution
First we show the tree-width of a forest is indeed one. Let's focus on the tree, we can of course decompose every tree independently. We make the following decomposition: decomposition tree has the same shape as the tree, root bag contains the root and every other node bag contains this node and its parent. One can easily check that this is correct.

For the other direction assume that $\mathrm{tw}(G) = 1$. We aim to show that there is no cycle in G. In there would be a cycle $C = \{v_1, \ldots, v_n, v_1\}$ then also $\mathrm{tw}(C) = 1$, so it is enough to show that tree-width of a cycle is bigger than one. Assume a decomposition of cycle C with tree width one. Some bag B_i contains edge $\{v_i, v_{i+1 \mod n}$ for every i. Clearly there is a path ρ_i from any bag B_i to bag $B_{i+1 \mod n}$, as node $v_{i+1 \mod n}$ belongs to both nodes. Definitely no two consecutive paths intersect. Therefore if we follow $\rho_1, \ldots, \rho_n$ we can a nontrivial loop of bags, contradiction with the fact that decomposition is a tree.

Remark: there might be a much simpler solution, this is the one we found on exercises.

Exercise 27.
Compute $\mathrm{tw}(K_n)$, where K_n is a clique of n vertices.

Solution
Of course $\mathrm{tw}(K_n) \leq n-1$, we can put all the nodes into one bag. We show now that there is no decomposition of tree width smaller or equal $n-2$, so containing at most $n-1$ nodes in one bag. Let us take the decomposition of tree width $\leq n-1$, but with minimal number of nodes. Consider some bag B, if we remove this bag set of nodes not being in this set separates into disjoint parts contained in different parts of decomposition tree. However, this cannot happen in K_n case, so it means that every node of decomposition tree has at most one neighbor. This means that there can be at most two nodes, which is impossible (we can easily check it).

Exercise 28.
We design the following game on graph G between k cops and a robber. Robber has a fast motorbike, cops have helicopters. In between moves everybody occupies one vertex. A move looks as follows:

- some subset of cops flies up and declares where they are going to land at the end of this move,
- robber moves, we cannot pass vertices, which are occupied by cops, which haven't flied up,
- cops land on the declared vertices.

Cops win if they land on a vertex, where the robber is. Assume that cops know where he is. Show that if $\mathrm{tw}(G) = k$ then $k+1$ cops have a winning strategy in this game. Remark: if $\mathrm{tw}(G) = k$ then the robber has a winning strategy against k cops, but this is harder to show.

Solution
Cops first choose a random bag U. Then they look where is the robber, i.e. in which part of the decomposition tree. It is easy to show that if $v \in U$ then v can only belong to bags in one direction in decomposition tree from U. Then cops move slowly towards this direction. Let U' be the first bag into this direction. Cops outside of $U \cap U'$ move to U'. Of course by having $\mathrm{tw}(G)+1$ cops we can place one at every vertex of U'. Then we follow the same strategy and finally robber will be caught.

Exercise 29.
Let G_k be a grid $k \times k$ (with k^2 vertices). Show that

- $\mathrm{tw}(G_k) \leq k$,
- $\mathrm{tw}(G_k) \geq k-1$.

Remark: $\mathrm{tw}(G_k) = k$, but showing $\mathrm{tw}(G_k) \geq k$ is a bit technical and I have not heard about any nice solution.

Solution
It is easy to see that $k+1$ cops will manage to catch the robber. They take the whole first row and then slowly move downwards such that they form a barrier without holes. At every moment they have the beginning of the row in row number i and the end in row number $i+1$, the place where there is a change of rows two cops stay in the same column.

Now we show that strategy of robber to avoid $k-1$ cops. If there are $k-1$ cops there is always some row and some column, where no cop is staying. This is the place which robber occupies. When cops change places we just moves to the new place. He first uses old-free row, to reach the new-free column and the new-free column to reach crossing of new-free column and new-free row.

Classes 8

Today we continue tree-width.

Exercise 30.
Determine $\mathrm{tw}(K_{n,n})$, where $K_{n,n}$ is a full bipartite graph with n vertices on the left and n vertices on the right.

Solution
We show that $\mathrm{tw}(K_{n,n}) = n$. We use one more time cops and robber game. It is easy to see that $n+1$ cops can catch robber. First n cops fly to all the vertices on one side. Then the last cop fly to the robber node and he cannot move anywhere.

On the other hand robber can avoid n cops. He just stays in his node. If some of the cops is approaching him it means that on the other half of the graph there is a free node. This is where robber moves in this round.

Exercise 31.
Show that vertex cover can be solved on graph G in time $2^{\mathcal{O}(\mathrm{tw}(G))} \cdot n^{\mathcal{O}(1)}$.

Solution
We do dynamic programming on the tree decomposition. It is actually easier to do this on nice tree decomposition, so such that all the vertices are either introduce, forget or join nodes or a leaf node. The updated information is as follows: for an already processed part of the graph and a current bag B we remember for every its subset $S \subseteq B$ how many vertices have to be taken into vertex cover under the condition that all the vertices from S are taken. It is quite easy to update this information on nice tree decomposition. Complexity is as need, as we remember for every of at most $2^{\mathrm{tw}(G)}$ nodes some small information.

We say that graph G is minor of graph H, denoted $G \trianglelefteq H$ if G can be obtained from H by a sequence one of three operations: 1) deleting a vertex, 2) deleting an edge, 3) contracting an edge, i.e. unifying two endpoints of this edge. Minor relation is a well known one. In particular there is a theorem that graph is planar iff K_5 or $K_{3,3}$ is not its minor.

Exercise 32.
Show that $G \trianglelefteq H$ implies $\mathrm{tw}(G) \leq \mathrm{tw}(H)$.

Solution
We have to show that none of three operations can enlarge tree width of a decomposition. For 1) and 2) it is trivial, for 3) it is also easy to see.

There are two useful theorems.

Theorem Let G_k be a grid with k^2 vertices. For every sufficiently big graph G if $\mathrm{tw}(G) \geq k^{99}$ then $G_k \trianglelefteq G$.

Theorem Let G_k be a grid with k^2 vertices. For every planar graph G is $\mathrm{tw}(G) \geq 5k$ then $G_k \trianglelefteq G$.

Exercise 33.
Show that for any $k \in \mathbb{N}$ there exists $t \in \mathbb{N}$ such that if graph G does not have k vertex disjoint cycles then $\mathrm{tw}(G) \leq t$.

Solution
We can put it in the other way: if $\mathrm{tw}(G) > t$ then G has k vertex disjoint cycles. Let N be constant such that for G with more than N vertices if $\mathrm{tw}(G) \geq k^{99}$ then $G_k \trianglelefteq G$. Let then $t = \max(N, k^{99})$, definitely for $\mathrm{tw}(G) \geq t$ we have $G_k \trianglelefteq G$. However in G_k there are k vertex disjoint cycles, so in G also, which finishes the proof.

Exercise 34.
Show that for a planar graph one can check in $2^{\mathcal{O}(\sqrt{k}\log(k))} \cdot n^{\mathcal{O}(1)}$ whether it contains a simple path with at least k vertices.

Solution
Assume that $k = m^2$ for some m (in the other case solution is very similar). If $G_m \trianglelefteq G$ then there is a simple path in G_m of length k, so the same in G. So by grid theorem for planar graphs if $\mathrm{tw}(G_m) \geq 5m$ then $G_m \trianglelefteq G$ and we are done. So one should just test whether $\mathrm{tw}(G_m) \geq 5m$ (which can be done in postulated time). It only remains to solve the case when $\mathrm{tw}(G_m) < 5m$. However in this case we have tree decomposition with tree width smaller than $5\sqrt{k}$. So then we can do dynamical algorithm on this decomposition. Here I will go into details of this algorithm. The above technique is much more general and is called bidimensionality.

Classes 9

Today we continue logic, MSO and ∃SO.

Exercise 35.
Show that the language of words of composite length is in ∃SO.

Solution
We guess the relation $+k$ such that length of the word is kn. We also guess the set of positions $0, k, 2k, (n-1)k$. It is easy to verify, that our relation R is of the form $+k$ for some k, we just have to check that $+1(+k(x)) = +k(+1(x))$ for every x ($+1$ is easy to implement using order). Then we check that the set is indeed of the postulated form, the first position is in the set and the last position in the set -1 and $+k$ is the last position in the word.

There are definitely another ways of solving this exercise.

This exercise is the special case of the more general fact (Fagin's theorem), which we will (maybe) show later.

Exercise 36.
Find a finite alphabet Σ and ϕ in MSO working on Σ-labelled graphs such that $\phi(G) =$ true iff G is a grid.

Solution
Alphabet will contain corner letters Σ_{cor} containing four different labels for corners, border letters Σ_{bor} containing four different letters for borders (top, bottom, left, right) and one inside letter Σ_{ins} for other nodes. Moreover this will be producted with alphabet for 0, 1 or 2 modulo 3 rows and columns. So every node will know what is its modulo in rows, in columns and whether it is on the corner, border or inside. Then we write in ϕ that corner and border nodes have appropriate neighbors and also that inside nodes have appropriate neighbors (from appropriate rows and columns). We also define for an edge whether it goes left, right, up or down (we can do it looking at numbers). Then we also define a predicate row and column for the set of nodes in one row or column. Then we write that for any vertical neighbor nodes their upper neighbors are also vertical neighbors and the same in all directions. And we also write that every row and column finishes at some moment (we have order, as we can go right or up for example). This should be sufficient to define grid.

Classes 10

Today we finish MSO and do a few exercises which use decidability of limitedness of distance automata.

Exercise 37.
Show that given ϕ in MSO it is undecidable whether it is satisfied in some finite graph.

Solution
We will reduce from the halting problem of a Turing Machine (TM). For a machine we will construct ϕ in MSO such that ϕ is satisfied if and only if Turing Machine halts. Actually ϕ will be true only in graphs which are grids of rectangle shape, which represent a finite run of TM.

We will say that graph is a lattice, as in the previous exercise. Then we say that two rows differ only near the head. All these things can be expressed in MSO.

Exercise 38.
Problem of *limitedness of $\mathcal{A}$ on regular language L* asks whether there exists $n \in \mathbb{N}$ such that for every word $w \in L$ the cost of w with respect to $\mathcal{A}$ is not bigger than n (i.e. $f_{\mathcal{A}}(w) \leq n$). Show that it is decidable.

Solution
We define distance automaton $\mathcal{A}'$ such that $\mathcal{A}'$ is limited on Σ^* (which is decidable due to the theorem) iff $\mathcal{A}$ is limited on L. It is simple, $\mathcal{A}'$ is $\mathcal{A}$ with

additional component, which has only costless transitions and accepts complement of language L.

Exercise 39.
Problem of *finite power property* asks whether for a given regular language L there exists $k \in \mathbb{N}$ such that $L^* = L^0 \cup L^1 \cup \ldots \cup L^k$. Show that this problem is decidable.

Solution
We will reduce to the problem of limitedness of distance automata on a given regular language. We take automaton for L and add costly transitions over every letter $a \in \Sigma$ from every accepting state to a state, which is reachable from initial state over a. These transitions correspond to starting new copy of L. One can easily verify that L has finite power iff the considered automaton is limited on L^*.

Exercise 40.
We say that languages K and L are *separated by* language S if $K \subseteq S$ and $L \cap S = \emptyset$. For $u, v \in \Sigma^*$ we say that $u = a_1 \cdots a_k \preceq v$ if $v \in \Sigma^* a_1 \Sigma^* \ldots \Sigma^* a_k \Sigma^*$. intuitively this means that u is a subsequence of v. Language L is *upward closed* if for every $u \in L$ and $u \preceq v$ we have $v \in L$. Show that deciding whether two given regular languages K and L can be separated by some upward closed language is decidable.

Solution
For a language K let $K^{\uparrow} = \{w \mid \exists_{u \in L} u \preceq w\}$. Let $\mathcal{UP}$ be the class of upward closed languages. We claim that K and L cannot be separated by some language from $\mathcal{UP}$ iff $K^{\uparrow} \cap L \neq \emptyset$. Start with right to left implication. Clearly $K^{\uparrow}$ is the smallest language from $\mathcal{UP}$ which contains K. We if $K^{\uparrow} \cap L \neq \emptyset$ then every language from $\mathcal{UP}$ which contains K also intersects L, which means that K and L cannot be separated. To show implication from left to right observe that if $K^{\uparrow} \cap L = \emptyset$ then $K^{\uparrow}$ is a good separator.

Exercise 41.
Let $\mathcal{F}$ be a class of finite unions of languages of the form $\Sigma^* w_1 \Sigma^* \ldots \Sigma^* w_k \Sigma^*$, where all w_i are words from Σ^*. Show that for given regular languages K and L it is decidable whether they can be separated by a set from $\mathcal{F}$.

Remark: Note that $\mathcal{F}$ is more general than $\mathcal{UP}$. To see this recall that Higman's Lemma implies that there is no infinite antichain in the $\preceq$ order. Therefore every upward closed language has finitely many minimal elements. Thus every upward closed language is a finite union of languages of the form $\Sigma^* a_1 \Sigma^* \ldots \Sigma^* a_k \Sigma^*$, where all $a_i \in \Sigma$.

Solution
Here we will use decidability of limitedness problem for distance automata. For K and L we build a distance automaton $\mathcal{A}_L$ (depending only on L, not on K) such that: K and L can be separated by a language from $\mathcal{F}$ iff automaton $\mathcal{A}_L$ is bounded on K. By decidability of limitedness problem it is enough how to construct such an automaton. For a word w automaton $\mathcal{A}_L$ will output the smallest number n such that there is a language $R = \Sigma^* w_1 \Sigma^* \ldots \Sigma^* w_k \Sigma^*$ with the property that $w \in R$, $R \cap L = \emptyset$ and sum of length of w_i equals n, so

$\sum_{i=1}^{k} |w_i| = n$. First we show that if $\mathcal{A}_L$ will indeed compute such a number it will fulfill the above condition. Assume that $\mathcal{A}_L$ computes such a number n. If $\mathcal{A}_L$ is limited on K this means that for every word $w \in K$ there is a short (not longer then n) expression which contains w, but does not touch L. However there is a finite number of such a short expressions. So if we take union of all of them not touching L we will obtain a separator from $\mathcal{F}$, which separates K and L. On the other hand if there is some separator $S \in \mathcal{F}$ it consists of finitely many expressions of the above form, such that every one does not touch L and every word from K belongs to at least one of these expressions. So if we take n to be maximal size among these expressions forming S we know that $\mathcal{A}_L$ outputs at most n on every word from K (so it is limited).

It therefore suffices to show how to construct $\mathcal{A}_L$ computing such a number n. $\mathcal{A}_L$ will guess sign by sign the optimal expression, from the left to the right. For every letter of the input word it will choose between two options: either we add a new letter to the constructed expression (this will be costly) or we do not add and the input letter is consumed by current Σ^* in the expression. All the time automaton has to remember which states of DFA of L can be reached by words belonging to currently constructed expressions. So there should be at least $2^{\text{size of DFA of } L}$ states of $\mathcal{A}_L$. Moreover $\mathcal{A}_L$ has to know whether in the expression we currently have a letter at the end of Σ^*, because in this second option it can process input letter without addition Σ^* and in the second not. This is all what $\mathcal{A}_L$ does, it finishes the proof.

Classes 11

Today we do mainly well quasi order technique (WQO). Let us recall the definition and main lemma. Here we assume we always consider orders, in general one can define it similarly on quasi-orders. Order $(X, \leq)$ is a well quasi order (WQO) if for any infinite sequence $x_1, x_2, \ldots$ of elements of X there exists a *dominating pair*, i.e indices $i < j$ such that $x_i \leq x_j$. The following lemma shows that there are two equivalent definitions.

Lemma
For any order $(X, \preceq)$ the following conditions are equivalent

1. in every infinite sequence $x_1, x_2, \ldots \in X$ there exist $i < j$ such that $x_i \preceq x_j$,
2. in every infinite sequence $x_1, x_2, \ldots \in X$ there exist an infinite subsequence $i_1 < i_2 < \ldots \in \mathbb{N}$ such that for $x_{i_j} \preceq x_{i_{j+1}}$ for all $j \in \mathbb{N}$,
3. there is no infinite descending sequence in X and no infinite antichain in X with respect to order $\preceq$.

Exercise 42.
Is the following order a WQO?

1. $\mathbb{N}^2$ with lexicographic order
2. words over $\{a, b\}$ with lexicographic order
3. $\mathbb{N}$ with divisibility order, i.e. $x \preceq y \iff x \mid y$

4. line segments with an order $[a, b] \preceq [c, d] \iff (b < c) \vee (a = c \wedge b \leq d)$
5. graphs with subgraph order
6. trees with subgraph order

Solution
Answers are the following.

1. Yes. Any pair, which is dominating in Dickson's order is also dominating in lexicographic order. So by Dickson's lemma lexicographic order is also a WQO.
2. No. An infinite descending sequence is of the form: $b, ab, aab, aaab, \ldots$.
3. No. Prime numbers are an infinite antichain.
4. Yes. There is no infinite descending sequence, because sum $a + b$ is decreasing. There is also no infinite antichain. Assume there is one. Let $[a, b]$ be an element of it. Any $[c, d]$ in the antichain has to have $c \leq b$. So there are finitely many options for c, so some two segments in the antichain are of the form $[c, d_1]$ and $[c, d_2]$. However they have to be comparable, contradiction.
5. No. Cycles C_n for $n \geq 3$ are an infinite antichain. The same works also for induced subgraph order.
6. No. An infinite antichain is formed by trees, which are paths of length n such that both end vertices have additionally two neighbors (all together three neighbors). The same example works for induced subgraph order.

Exercise 43.
Show that if $(X, \leq_X)$ and $(Y, \leq_Y)$ are both WQOs then also $(X \times Y, \leq)$ is WQO, where $(x, y) \leq (x', y') \iff x \leq_X x' \wedge y \leq_Y y'$.

Solution
Consider an infinite sequence of elements of $X \times Y$. By the fact that $\leq_X$ is WQO there exists an infinite subsequence such that first coordinates form an increasing subsequence. Then in that subsequence by the fact that $\leq_Y$ is WQO there exists a dominating pair on second coordinates. This pair is thus also a dominating pair in the order $\leq$.

Exercise 44.
Show that there exist a polynomial algorithm deciding whether a given graph is planar. Hint: assume that there exist a polynomial algorithm deciding whether a given graph G is a minor of an input graph H.

Solution
We use the Wagner (or Kuratowski) theorem that H is planar iff neither K_5 nor $K_{3,3}$ is its minor. By Hint above we can easily check it in PTIME.

Exercise 45.
Show that there exist a polynomial algorithm deciding whether a given graph can be drawn on torus without crossing edges.

Solution
It is not hard to observe that if $H \trianglelefteq H'$ and H' can be drawn on torus then also H can be drawn on torus. Therefore set of graphs which can be drawn on torus is downward closed in the minor order $\trianglelefteq$. Thus its complement is upward closed. By Robertson-Seimour theorem we know that $\trianglelefteq$ is WQO. Set of minimal (wrt. to $\trianglelefteq$) graphs, which cannot be drawn on torus is an antichain, thus it is finite set, say these are $G_1, \ldots, G_k$. Therefore H can be drawn on torus iff none of G_i is a minor of H. Therefore it is enough to check whether all these G_i are minors on the input graph, which is in PTIME by hint above. Note that we know that such an algorithm exist, but we do not know graphs G_i, so we do not know how exactly this algorithm looks like.

Exercise 46.
Show that given a vector addition system (VAS) $V = (s, T)$ it is decidable whether its reachability set Reach(V) is finite.

Solution
We build a tree with root being vector s and children of every vector v being all the $v+t$ for $t \in T$ such that $v+t \in \mathbb{N}^d$. However we can this tree in the following way. If there is some vertex $v \in \mathbb{N}^d$ such that there exists its ancestor vertex $u \in \mathbb{N}^d$ with $u \preceq v$ then we do not continue expanding vertex v. There are two cases. If v is strictly bigger than u on some coordinate by detecting dominating pair (u, v) on this path we know that reachability set is infinite. In the other case, if $u = v$, we know that it makes no sense to expand this path, because we will not reach anything new. By Dickson's lemma we know that every path is finite. Tree is finitely branching, therefore by Königs lemma the whole tree is finite. Therefore at some moment we will compute the whole tree an algorithm will be finished. If all dominating pairs where $u = v$ then reachability set is finite, otherwise it is infinite.

Exercise 47.
(Infinite Ramsey Theorem) Show that is we color every edge of an infinite (countable) clique into one of finitely many colors then there always exist a monochromatic clique.

Solution
We sort vertices from left to right. First vertex has infinitely many outgoing edges, at least one color appears infinitely many times. We choose such a color an leave only neighbors of this first vertex v_1 which have such colored edge to v_1. Then we take v_2 (in the filtered sequence), there also exists a color such that v_2 has infinitely many neighbors (to the right) with this color. We one more time filter vertices to the right of v_2 leaving only these which have appropriately colored edge with v_2. In that way we also define $v_3, v_4, \ldots$. We always keep already defined vertices to the left untouched. In that way we define v_k for every $k \in \mathbb{N}$ so we have an infinite sequence of vertices v_i. Every one has a distinguished color, so there exists a color in which there are infinitely many vertices. They form a monochromatic clique.

Exercise 48.
Consider a system in which we start from a fixed word $w \in \Sigma^*$. We also have a finite set T of rewriting rules of the form $u_i \mapsto v_i$, where $u_i, v_i \in \Sigma^*$. If there is

a rule $u_i \mapsto v_i$ in T then we can make a move $su_is' \to sv_is'$. In every moment we can also forget about any letter, so we can make a move $uav \to uv$ for any $a \in \Sigma$. Show that it is decidable for a given word u whether it is reachable from the start word w by finitely many moves.

Solution
TODO.

Classes 12

After some classes by Mikołaj I did one classes about transducers. I write here exercises without solutions. I use notation W for class of functions recognizable by DFA with oracles (in polish "Wyrocznia") and RW functions recognizable by DFA with registers and oracles and R functions recognizable by DFA with registers ($RW = R$, but this is not assumed in all the exercises).

Exercise 49.
Show that reversing the word does not belong to W.

Exercise 50.
Does $a^n \mapsto a^{n^2}$ belong to W?

Exercise 51.
Does $a^n \mapsto a^{\sqrt{n}}$ belong to W?

Exercise 52.
Show that $W \circ W = W$.

Exercise 53.
Show that $RW = \text{compute}_r \circ W$.

Exercise 54.
Show that $RW = R \circ W$ and that $RW = RW \circ W$.

Exercise 55.
Show that for function $f : \Sigma^* \to \Gamma^*$ and regular language $L \subseteq \Gamma^*$ the language $f^{-1}(L)$ is regular a) for $f \in W$, b) for $f \in RW$.