Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Philipp Bamberger, Yannic Maus

Theoretical Computer Science - Bridging Course
Summer Term 2018
Exercise Sheet 9

for getting feedback submit (electronically) before the start of the tutorial on 7th of January 2019.

Exercise 1: The class $\mathcal{NPC}$ *(8 Points)*

As this type of exercise is really important for the course we have another one.

A subset of the nodes of a graph G is a **dominating set** if every other node of G is adjacent to some node in the subset. Let

$$\textsc{DominatingSet} = \{\langle G, k\rangle \mid \text{has a dominating set with } k \text{ nodes}\}.$$

Show that DOMINATINGSET is in $\mathcal{NPC}$. Use that

$$\textsc{VertexCover} := \{\langle G, k\rangle \mid \text{ Graph } G \text{ has a } \textit{vertex cover} \text{ of size at most } k\} \in \mathcal{NPC}\ .$$

Remark: A VERTEXCOVER *is a subset $V' \subseteq V$ of nodes of $G = (V, E)$ such that every edge of G is adjacent to a node in the subset.*

Sample Solution

DominatingSet $\in \mathcal{NP}$: It is easy to show that Dominating Set is in NP. To test whether $\langle G, k\rangle \in \textit{DominatingSet}$ do the following guess and check procedure.
Guess: Guess a subset $D \subseteq V$ of the nodes of size k.
Check: Given a subset $D \subseteq V$ of the nodes, one can verify in polynomial time if that is a dominating set. This can be done by taking each vertex $v \in V$ and checking if either $v \in D$ or one of its edges travel into the set.

DominatingSet is $\mathcal{NP}$-hard: To show that is $\mathcal{NP}$-complete, first of all notice that a dominating set has to include all isolated vertices (those which have no edges from them). So let us assume that our graph does not have any isolated vertices. We will show that Dominating Set is NP-complete using a reduction from Vertex Cover. Given a graph G for which we have to check containment in VERTEXCOVER, we will construct a graph G' as follows: G' has all edges and vertices of G. Also, for every edge $\{u, v\} \in E(G)$, we add an addition node w and the edges $\{u, w\}$ and $\{w, v\}$ in G'. Now we will show that G has a vertex cover of size k if and only if G' has a dominating set of the same size. If S is a vertex cover in G, we will show that S is a dominating set for G'. S is a vertex cover implies every edge in G has at least one of its end points in S. Consider $v \in G'$. If v is an original node in G, then either $v \in S$ or there must be some edge connecting v to some other vertex u. Then if $v \notin S$, u must be in S, and hence there is an adjacent vertex of v in S. So v is covered by some element in S. However, if w is an additional node in G', then w has two adjacent vertices $u, v \in G$ and using the

above argument at least one of them is in S. So the additional nodes are also covered by S. So if G has a vertex cover, then G' has a dominating set of same size (in fact the same set itself would do). If G' has a dominating set D of size k, then look at all the additional vertices $w \in D$. Notice that w must be connected to exactly 2 vertices $u, v \in G$. Now see that we can safely replace w by one of u or v. w in D dominates only $u, v, w \in G'$. But these three edges form a clique, and we can as well pick u or v and still dominate all the vertices that w used to dominate. So we can eliminate all the additional vertices as above. Since all the additional vertices correspond to one of the edges in G, and since all of the additional vertices are covered by the modified D, this means that all the edges in G are covered by the set. So if G' has a dominating set of size k, then G has a vertex cover of size at most k.
We showed that a dominating set of size k exists in G' if and only if a vertex cover of size k exists in G. Since we know that vertex cover is an $\mathcal{NP}$-complete, Dominating Set is also $\mathcal{NP}$-complete.

Hint: Go through similar exercises from previous years (and the internet) to study for the exam.

Exercise 2: Complexity Classes: Big Picture *(2+3+2 Points)*

(a) Why is $\mathcal{P} \subseteq \mathcal{NP}$?

(b) Show that $\mathcal{P} \cap \mathcal{NPC} = \emptyset$ if $\mathcal{P} \neq \mathcal{NP}$.
Hint: Assume that there exists a $L \in \mathcal{P} \cap \mathcal{NPC}$ and derive a contradiction to $\mathcal{P} \neq \mathcal{NP}$.

(c) Give a Venn Diagram showing the sets $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}$ for both cases $\mathcal{P} \neq \mathcal{NP}$ and $\mathcal{P} = \mathcal{NP}$.
Remark: Use the results of (a) and (b) even if you did not succeed in proving those.

Sample Solution

(a) If $L \in \mathcal{P}$ there is a deterministic Turing machine that decides L in polynomial time. Then $L \in \mathcal{NP}$ simply by definition since a deterministic Turing machine is a special case of a non-deterministic one.

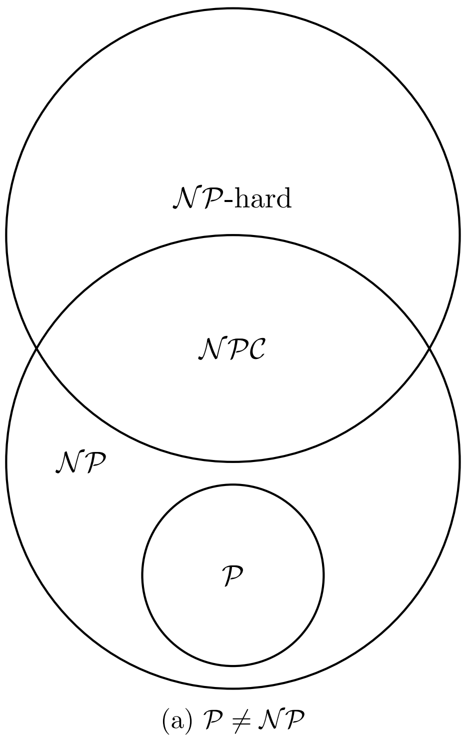
(b) As the hint suggests we assume that there is a language L which is $\mathcal{NP}$-complete and simultaneously solvable in polynomial time by a Turing machine. We use this language L to show that $\mathcal{NP} \subseteq \mathcal{P}$, which together with (a) implies $\mathcal{NP} = \mathcal{P}$, i.e., a contradiction to our premise $\mathcal{NP} \neq \mathcal{P}$. Hence L cannot exist if $\mathcal{NP} \neq \mathcal{P}$.

So let $L' \in \mathcal{NP}$. We want to show that L' is in $\mathcal{P}$ to obtain the contradiction. Since L is also $\mathcal{NP}$-hard, we can solve the decision problem L' via L by using the polynomial reduction $L' \leq_p L$. In particular for any string $s \in L'$ we have the equivalency $s \in L' \iff f(s) \in L$, where f is induced by the reduction.

We construct a Turing machine for L' that runs in poly. time. For instance s it first computes $f(s)$ in polynomial time and then uses the Turing machine for L as a subroutine to return the answer of $f(s) \in L$ in polynomial time. In total, we require only polynomial time to decide $s \in L'$ which means $L' \in \mathcal{P}$.

(c) See Figure 1. For the case $\mathcal{P} = \mathcal{NP}$, the notion of $\mathcal{NP}$-hardness becomes utterly meaningless since the class $\mathcal{NP}$ can be polynomially reduced to every other language except Σ^* and $\emptyset$. In order to show that $L' \leq_p L$ for an $L \neq \Sigma^*, \emptyset$ and for all $L' \in \mathcal{NP} = \mathcal{P}$, we need show that there is a polynomially computable mapping f such that $\forall s \in \Sigma^*\colon\ s \in L' \Leftrightarrow f(s) \in L$.

But such a mapping f always exists for $L \neq \Sigma^*, \emptyset$. We simply have to use a known 'yes-instance' $y \in L$ and a 'no-instance' $n \notin L$. Then we define for $s \in \Sigma^*$ that $f(s) := y$ if $s \in L'$ and $f(s) := n$ if $s \notin L'$. This obviously fulfills the above equivalency. Moreover f is polynomially computable since we can find out whether $s \in L'$ in polynomial time.



(a) $\mathcal{P} \neq \mathcal{NP}$

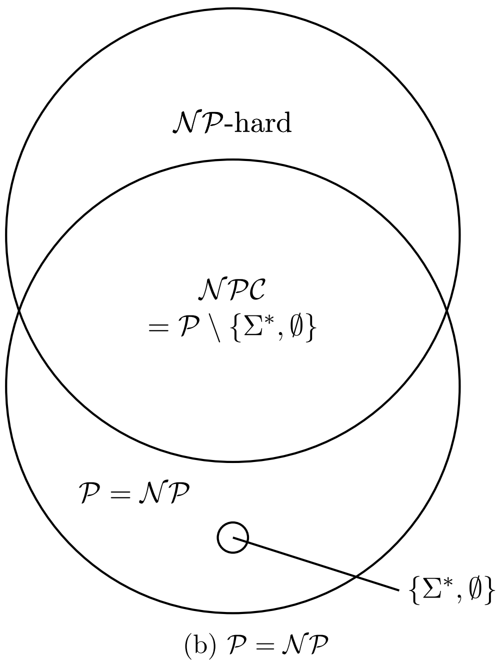

$\mathcal{NP}$-hard
$\mathcal{NPC}$
$= \mathcal{P} \setminus \{\Sigma^*, \emptyset\}$
$\mathcal{P} = \mathcal{NP}$
$\{\Sigma^*, \emptyset\}$
(b) $\mathcal{P} = \mathcal{NP}$

Figure 1: Venn-Diagram of the Language classes $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}, \mathcal{NP}$-hard.

Exercise 3: The class $\mathcal{P}$ *(1+2+2+1 Points)*

CLIQUE:

- A *clique* of a graph $G = (V, E)$ is a subset $Q \subseteq V$ such that for all $u, v \in Q : \{u, v\} \in E$.
- **Input**: Encoding $\langle G, k\rangle$ of an undirected, unweighted, simple graph $G = (V, E)$ and $k \in \mathbb{N}$.
- **Question**: Is there a clique of size at least k?

$\mathcal{P}$ is the set of languages which can be decided by an algorithm whose runtime can be bounded by $p(n)$, where p is a polynomial and n the size of the respective input (problem instance). Show that the following languages ($\cong$ problems) are in the class $\mathcal{P}$. Since it is typically easy (i.e. feasible in polynomial time) to decide whether an input is well-formed, your algorithm only needs to consider well-formed inputs. Use the $\mathcal{O}$-notation to bound the run-time of your algorithm.

(a) 1-DOMINATINGSET

(b) 2-VERTEXCOLORING

(c) 3-CLIQUE

(d) Any context-free language L. *Hint: You can use results from previous exercise sheets.*

Sample Solution

(a) A dominating set of size one exists if and only if the input graph $G = (V, E)$ has a node with degree $|V| - 1$. Thus one possible algorithm that decides 1-DOMINATINGSET checks for each node of G if it dominates all others. For this purpose we traverse all edges in the input graph G and maintain a count of the degree for all nodes. Then we check whether there is a node with maximum degree $|V| - 1$. If the result is positive we accept. Otherwise we reject $\langle G\rangle$. Since edges and nodes are traversed only a constant number of times the runtime of the algorithm is $\mathcal{O}(|E| + |V|) \subseteq \mathcal{O}(|\langle G\rangle|)$ ($|\langle G\rangle|$ is the length of the encoding of G). Therefore 1-DOMINATINGSET $\in \mathcal{P}$

(b) The following algorithm checks whether a connected component of a graph can be colored with two colors. Since the following algorithm can simply be repeated for every component of the input graph $G = (V, E)$ we assume that G is connected. We execute a Breadth First Search (BFS) on G starting from an arbitrary node $s \in V$. BFS is a very basic algorithm to traverse all nodes of a graph and uses a queue. It works as follows.

Initially we color s with the color 1 and enqueue all its neighbors while setting s as their predecessor. Then we repeat the following steps as long the queue is not empty. Take the first node v from the queue, color v with the color its predecessor is *not* colored in. Then check whether the color of one of v's *colored* neighbors equals v's own color and if this is the case, reject G and terminate. Otherwise enqueue all of v's *uncolored* neighbors, set v as their predecessor and continue. Finally, if the queue runs empty, G is accepted.

A BFS explores the graph in layers, i.e. it first explores all nodes that can be reached via a single edge (hop) from the start node s then all nodes with distance of exactly two hops from s and so on. If $\langle G \rangle \in$ 2-VertexColoring can be colored using only two colors, then the coloring is fixed by assigning an initial color to a single node s. This initial coloring of s is subsequently completed by the BFS, which colors the layers around s with alternating colors.

If $\langle G \rangle \notin$ 2-VertexColoring, i.e. no 2-coloring exists for G, then the BFS (which explores all edges) finds an edge with equally colored end nodes and rejects G. Since we explore every edge at most twice and every node enters and leaves the queue exactly once, the run-time is $\mathcal{O}(|V| + |E|)$ and thus polynomial in $|\langle G \rangle|$. If G consists of several components, we restart the algorithm with an arbitrary, uncolored node every time the queue runs empty, as long as uncolored nodes exist. This does not change the asymptotic run-time bound. Thus 2-VertexColoring $\in \mathcal{P}$.

(c) Let $G = (V, E)$ and $|V| = n$. Then we know $|E| = \mathcal{O}(n^2)$. Upon input G, we can enumerate all possible triples (v_1, v_2, v_3) such that $v_1 \neq v_2 \neq v_3 \neq v_1$. There exist at most $\binom{n}{3} = \mathcal{O}(n^3)$ such triples. For each such triple (v_1, v_2, v_3), we examine whether $(v_1, v_2) \in E$, $(v_1, v_3) \in E$, and $(v_2, v_3) \in E$. Since $|E| = O(n^2)$, this examination can be done in $O(n^2)$ time. If during the examination process, we find one triple that satisfies the requirement, we found a clique of size 3 accept G. Otherwise, when we finish examining all possible triple, we reject G since it does not contain a clique of size 3. The runtime of the above procedure is $\mathcal{O}(n^5)$, thus 3-Clique $\in \mathcal{P}$.

(d) Every context free language L has a context-free grammar G (definition) which can wlog assumed to be in Chomsky Normal Form (CNF). Unfortunately it was almost impossible to solve this exercise with the material from the course. So see it as an optimal question. In previous years we introduces the Cocke-Younger-Kasami (CYK) algorithm which solves the word problem ($s \in L(G)$?) for grammar G in CNF and any input string $s \in \Sigma^*$ in $\mathcal{O}(|s|^3)$. Thus $L \in \mathcal{P}$.