Repetition of Course Material

Let $L_1, L_2$ be languages (problems) over alphabets $\Sigma_1, \Sigma_2$. Then $L_1 \leq_p L_2$ ($L_1$ is polynomially reducible to $L_2$), iff a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ exists, that can be calculated in polynomial time and

$$\forall s \in \Sigma_1 : s \in L_1 \iff f(s) \in L_2.$$ 

Language $L$ is called $\mathcal{NP}$-hard, if all languages $L' \in \mathcal{NP}$ are polynomially reducible to $L$, i.e.

$$L \ \mathcal{NP}\text{-hard} \iff \forall L' \in \mathcal{NP} : L' \leq_p L.$$ 

The reduction relation '$\leq_p$' is transitive ($L_1 \leq_p L_2$ and $L_2 \leq_p L_3 \Rightarrow L_1 \leq_p L_3$). Therefore, in order to show that $L$ is $\mathcal{NP}$-hard, it suffices to reduce a known $\mathcal{NP}$-hard problem $\tilde{L}$ to $L$, i.e. $\tilde{L} \leq_p L$.

Finally a language is called $\mathcal{NP}$-complete ($\iff$), if

1. $L \in \mathcal{NP}$ and
2. $L$ is $\mathcal{NP}$-hard.

Exercise 1: The class $\mathcal{NP}\mathcal{C}$

This exercise is really (!!) important for the course.

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

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

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

$$\text{VERTEXCOVER} := \{\langle G, k \rangle \mid \text{Graph } G \text{ has a vertex cover of size at most } k\} \in \mathcal{NP}\mathcal{C}.$$ 

Remark: A vertex cover 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

$\text{DOMINATINGSET} \in \mathcal{NP}$: It is easy to show that Dominating Set is in NP. To test whether $\langle G, k \rangle \in \text{DOMINATINGSET}$ 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 \( \mathcal{NP} \)-complete using a reduction from Vertex Cover. Given a graph \( G \) for which we have to check containment in \( \text{VERTEXCOVER} \), 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: \( P \) and \( \mathcal{NP} \)? \( (3 + (2 + 3*) \) Points)\)**

Let \( \text{CNF}_k = \{ (\phi) \mid \phi \text{ is a satisfiable cnf-formula where each variable appears in at most } k \text{ places} \} \).

(a) Assume that \( P \neq \mathcal{NP} \) holds. Decide whether \( \text{CNF}_2 \) is in \( P \) or in \( \mathcal{NP} \setminus P \). Prove your claim!

(b) Show that \( \text{CNF}_3 \) is \( \mathcal{NP} \)-complete.

**Remark:** You can gain 3 additional points in this exercise to pass the 50% barrier.

**Sample Solution**

(a) On input \( \phi \), a turing machine \( M \) does the following for each clause in \( \phi \).

(i) If there are single literal clauses of the forms \( x \) and \( \neg x \) in \( \phi \), then reject i.e., \( \phi \) is not satisfiable.

(ii) Otherwise, consider clause of the form \( x \lor A \). If \( x \) does not appear negated in the other clauses, remove every clause of the form \( x \lor A \) from \( \phi \) by assigning true value at \( x \) and call the new formula as \( \phi \). If there remain no clause in \( \phi \), accept.

(iii) If there are two clauses of the form \( x \lor A \) and \( \neg x \lor B \) in \( \phi \), replace them by \( A \lor B \) in \( \phi \).

Repeat the above three processes until you get clauses with all different variables. Then assign all the literals true values and accept \( \phi \).

(b) First we show \( \text{CNF}_3 \) is in \( \mathcal{NP} \): given an assignment, one can verify in polynomial time whether formula \( \phi \) is satisfiable. The assignment of values of the variables is chosen non-deterministically. To show \( \text{CNF}_3 \) is \( \mathcal{NP} \)-hard, we show a reduction from \( 3\text{SAT} \). That is given an instance of the \( 3\text{SAT} \) problem, we convert it to an instance of a \( \text{CNF}_3 \) problem. Suppose \( \phi \) is a given \( 3\text{SAT} \)
instance. Let some variable occur \( k \) times in \( \phi \), where \( k > 3 \). Replace these \( k \) occurrences by new variables \( x_1, x_2, \ldots, x_k \), where each occurrence is replaced by one of these variables. Now for any interpretation of the input 3SAT formula \( \phi \), these new variables have to be assigned some truth values such that \( x_1 \equiv x_2 \equiv \cdots \equiv x_k \) i.e., \((x_1 \rightarrow x_2) \land (x_2 \rightarrow x_3) \land \cdots \land (x_{k-1} \rightarrow x_k) \land (x_k \rightarrow x_1) \) i.e., \((\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \cdots \land (\neg x_k \lor x_1) \). Thus each of the new variables occur exactly three times because \( x_i \) occurs once in place of the original \( i \)-th occurrence and twice in the new clauses added. Clearly, the length of the transformed formula is \( O(|\phi|) \) and \( \phi \) is satisfiable iff the transformed formula is.

**Exercise 3: Complexity Classes: Big Picture**

(2+3+2 Points)

(a) Why is \( P \subseteq NP \)?

(b) Show that \( P \cap NP = \emptyset \) if \( P \neq NP \).

**Hint:** Assume that there exists a \( L \in P \cap NP \) and derive a contradiction to \( P \neq NP \).

(c) Give a Venn Diagram showing the sets \( P, NP, P \cap NP \) for both cases \( P \neq NP \) and \( P = NP \).

**Remark:** Use the results of (a) and (b) even if you did not succeed in proving those.

**Sample Solution**

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

So let \( L' \in NP \). We want to show that \( L' \) is in \( \mathcal{P} \) to obtain the contradiction. Since \( L \) is also \( NP \)-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 ??_. For the case \( \mathcal{P} = NP \), the notion of \( NP \)-hardness becomes utterly meaningless since the class \( NP \) can be polynomially reduced to every other language except \( \Sigma^* \) and \( \emptyset \). In order to show that \( L' \leq_p L \) for an \( L \neq \Sigma^*, \emptyset \) and for all \( L' \in NP = \mathcal{P} \), we need show that there is a polynomially computable mapping \( f \) such that \( \forall s \in \Sigma^* : s \in L' \iff f(s) \in L \).

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.

3
Figure 1: Venn-Diagram of the Language classes $\mathcal{P}, \mathcal{NP}, \mathcal{NPC}, \mathcal{NP-hard}$.