Tutorial 11

Exercise 1 (compulsory)

Is the following definition of NP-completeness correct? If no, give arguments why not.

Definition: A Turing machine M is NP-complete if $M \in$ NP and for every $L \in$ NP we have $L \leq_P M$.

Solution:
The definition does not make any sense. The elements in the class NP are languages (sets of strings), not Turing machines (neither the encodings of Turing machines). Hence M, nor $\langle M \rangle$ cannot be a member of NP.

Exercise 2 (compulsory)

Consider the following claims and answer which of them are correct and which of them not. Give precise arguments.

1. If L_1 and L_2 are NP-complete, then $L_1 \leq_P L_2$ and $L_2 \leq_P L_1$.
2. If L_1 and L_2 are NP-complete, then $L_1 \leq_m L_2$ and $L_2 \leq_m L_1$.
3. If $L_1 \leq_P L_2$, $L_2 \leq_P L_1$, and $L_1, L_2 \in$ NP, then L_1 and L_2 are both NP-complete.
4. If L_1 is NP-complete and $L_1 \leq_P L_2$, then L_2 is NP-complete.

Solution:

1. Correct. It follows directly from the definition of NP-completeness (part 2 about NP-hardness).
2. Correct. Surely, if $L_1 \leq_P L_2$ then also $L_1 \leq_m L_2$, and if $L_2 \leq_P L_1$ then also $L_2 \leq_m L_1$. This is because every polynomial time reduction is also a mapping reduction.
3. Incorrect. Let $L_1 = L_2 = \emptyset$. Surely $\emptyset \leq_P \emptyset$ by a reduction that is e.g. the identity function but $\emptyset$ cannot be NP-complete (because none of the langauges in NP, except for $\emptyset$ itself, are reducible to $\emptyset$).
4. Incorrect. We know that L_2 is NP-hard but we have no guarantee that L_2 belongs to NP, which is required for NP-completeness. The correct claim is: If L_1 is NP-complete, $L_2 \in$ NP, and $L_1 \leq_P L_2$ then L_2 is NP-complete.

Exercise 3 (compulsory)

Consider a nondeterministic Turing machine M over the input alphabet $\Sigma = \{a\}$, tape alphabet $\Gamma = \{a, \sqcup\}$, control states $Q = \{q, q_{accept}, q_{reject}\}$ and the following δ function:

- $\delta(q, a) = \{(q, a, R), (q_{reject}, \sqcup, L)\}$
- $\delta(q, \sqcup) = \{(q_{accept}, a, R)\}$

Fill in all legal windows (recall the proof of Cook-Levin Theorem) with the following chosen first rows. Note that this covers only some of the legal windows. The number of windows in every row indicates how many possible legal windows you should be able to find. You might use the abbreviations q_a for q_{accept} and q_r for q_{reject}.

a	a	a

a	a	a

a	a	a

a	a	a

#	a	a

#	a	a

a	q	a

a	q	a

a	q	⊔

q	a	a

q	a	a

q	a	a

q	a	a

a	a	q

a	a	q

Solution:

a	a	a
a	a	a

a	a	a
q	a	a

a	a	a
⊔	a	a

a	a	a
a	a	q_r

#	a	a
#	a	a

#	a	a
#	a	q_r

a	q	a
a	a	q

a	q	a
q_r	a	⊔

a	q	⊔
a	a	q_a

q	a	a
a	q	a

q	a	a
a	⊔	a

q	a	a
q_r	⊔	a

q	a	a
⊔	⊔	a

a	a	q
a	a	a

a	a	q
a	q_r	a

Exercise 4 (compulsory)

Show that if P=NP then every language $B \in$ P, except for $\emptyset$ and Σ^*, is NP-complete.

Solution:
Assume that P=NP. Let $B \in$ P such that $B \neq \emptyset$ and $B \neq \Sigma^*$. This means that there is a string $w_{in} \in B$ and a string $w_{out} \notin B$. We want to show that B is NP-complete.

Surely, the language B is in NP=P (by our assumption). We have to show that B is NP-complete. Let A be an arbitrary language from NP=P. Hence A has a polynomial time decider M_A. We need to argue that $A \leq_P B$. Here is a poly-time reduction f from A to B:

> "On input w:
> 1. Run M_A (decider for A) on w.
> 2. If M_A accepted then output w_{in}. If M_A rejected then output w_{out}."

This is a poly-time reduction from A to B, and hence B is NP-complete.

Exercise 5 (optional)

Define the language

$$A_{BTM} = \{\langle M, x, 1^t\rangle \mid M \text{ is a NTM which accepts the input } x \text{ after no more than } t \text{ steps}\}$$

where 1^t here denotes the number t written in unary, i.e., the string $\underbrace{1 \ldots 1}_{t \text{ occurrences}}$. Prove that A_{BTM} is NP-complete. *Hint:* Make a direct proof, similarly as Cook-Levin Theorem.

Solution:
We must show that

- $A_{BTM} \in$ NP, and
- for all $L \in$ NP we have $L \leq_P A_{BTM}$.

To show that A_{BTM} is in NP, we construct a nondeterministic polynomial-time decider for A_{BTM}. Here it is:

"On input $\langle M, x, 1^t\rangle$:

1. Nondeterministically simulate a computation of M on x for t steps.
2. If M accepted x, then accept else reject."

If $n = |\langle M, x, 1^t\rangle|$, then it is easy to see that the decider uses $O(t) \cdot O(n) = O(n^2)$ steps.

We must now show that for every $L \in$ NP we have $L \leq_P A_{BTM}$. So we must find a polynomial-time computable function f with the property that $w \in L \iff f(w) \in A_{BTM}$.

Since $L \in$ NP, we know that there exists a nondeterministic Turing machine M_L with time complexity $O(n^k)$ which is a decider for L. We thus know that for any string w we have that $w \in L$ if and only if M accepts w in $c \cdot n^k$ of less steps for some constant c. So let

$$f(w) = \langle M, w, 1^{c \cdot |w|^k}\rangle$$

This function clearly satisfies that $w \in L \iff f(w) \in A_{BTM}$. We now only need to show that f is polynomial-time computable.

It is not hard to show that we can compute the unary representation of the number $c \cdot n^k$ in time polynomial in n. A two-tape Turing machine for computing the unary representation of $c \cdot n^k$ given the unary representation of n looks as follows:

"On input 1^n:

1. Make a copy of n on the second tape.
2. For $i := 1$ to k
   - Create n copies of the string on the second tape.
3. Finally, create c copies of the string that is on the second tape."

Stage 1 requires $O(n)$ steps. The loop in stage 2 is traversed a total of $O(n)$ times. Each traversal makes n copies of a string whose length is at most $O(n^k)$; this requires no more than $O(n \cdot n^{2k}) = O(n^{2k+1})$ steps. Consequently, in the step 2. the machine uses at most $O(n) + nO(n^{2k+1}) = O(n^{2k+2})$ steps. Finally, step 3. uses $O(n^{2k})$ steps. Hence the total running time is polynomial (by Theorem 7.8, we can simulate the two-tape machine by a single-tape machine with only quadratic increase in the time complexity).