**Technische Universität München (I7)**
**Prof. J. Křetínský / Dr. C.H. Broadbent**

**Summer term 2016**
**3 May 2016**

# Solution

## Computational Complexity – Homework 4

Discussed on 9 May 2016.

### Exercise 4.1

Let us denote $\mathbf{DP} = \{L \mid \exists M, N \in \mathbf{NP} : L = M \setminus N\}$ the class of languages that are diferences of two NP languages.

(a) Show that $C = \{\langle G_1, k_1, G_2, k_2\rangle \mid G_1$ has a $k_1$-clique and $G_2$ does not have any $k_2$-clique$\}$ is DP-complete.

(b) Show that $MAX - CLIQUE = \{\langle G, k\rangle \mid$ the largest clique of $G$ is of size exactly $k\}$ is DP-complete.

(c) It is unknown whether $MAX - CLIQUE$ is in $\mathbf{NP}$. Show that if $\mathbf{P} = \mathbf{NP}$ then $MAX - CLIQUE$ is in $\mathbf{NP}$ and a largest clique can be found in polynomial time.

**Solution:**

(a) Let us define

$$C_1 := \{\langle G_1, k_1, G_2, k_2\rangle \mid G_1 \text{ has a } k_1\text{-clique and } G_2 \text{ is any graph and } k_2 \geq 0\}$$

and

$$C_2 := \{\langle G_1, k_1, G_2, k_2\rangle \mid G_2 \text{ has a } k_2\text{-clique and } G_1 \text{ is any graph and } k_1 \geq 0\}$$

Both these languages are NP-complete as we have already seen (they are both just the clique problem with unconstrained extra data attached to each)

Then $C = C_1\ C_2$ and so $C \in DP$.

Now we show that $C$ is $DP$-hard. Suppose that $L \in DP$. It must be the case that $L = L_1\ L_2$ where $L_1, L_2 \in NP$ (by definition). By the NP-completeness of the clique problem, we must thus have polynomial-time reductions $f_1$ and $f_2$ from $L_1$ and $L_2$ to respectively to the clique problem.

Thus we can define a polynomial-time reduction $f$ from $L$ to $C$ by:

$$f(w) := \langle f_1(w), f_2(w)\rangle$$

This is clearly computable in polynomial time (since both $f_1(w)$ and $f_2(w)$ are), and moreover we have $w \in L$ iff $f(w) \in C$ since $w \in L$, iff $w \in L_1$ and $w \notin L_2$, iff $f_1(w) \in CLIQUE$ and $f_2(w) \notin CLIQUE$ (since $f_1$ and $f_2$ are reductions to $CLIQUE$), iff $f(w) \in C$.

(b) To see that MAX-CLIQUE is in $DP$, observe that the problem $CLIQUE+ := \{\langle G, k\rangle \mid G$ has a $k+$ 1clique$\}$ is in NP. It is then the case that MAX-CLIQUE is equal to $CLIQUE\ CLIQUE+$.

To show that MAX-CLIQUE is DP-hard, we show that there is a polynomial time reduction from $C$ (in the previous part) to MAX-CLIQUE.

Consider a tuple $\langle G_1, k_1, G_2, k_2\rangle$. We define a pair $\langle G, k\rangle$ that can be computed from the tuple in polynomial time such that $\langle G_1, k_1, G_2, k_2\rangle \in C$ iff $\langle G, k\rangle \in MAX - CLIQUE$.

Let $N_1$, $N_2$ be respectively the node sets of $G_1$ and $G_2$ and $E_1, E_2$ their respective edge relations. Let $K_{k'}$ be the clique of size $k'$.

Consider first the graph $G_1' := (N_1', E_1')$, where $N_1' := [1, k_1] \times N_1$ and $E_1' := \{((i, u), (i+1, v) \mid i \in [1, k_1)$ and $(u, v) \in E_1\}$. By construction, no clique of $G_1'$ can be bigger than $k_1$, and it will have a clique of size $k_1$ iff $G_1$ also has a clique of size $k_1$.

For $r \in \mathbb{N}$, we extend $G_1'$ to a graph $G_1^r$ by adding an instance of $K_r$ and then an edge from each node in this instance of $K_r$ to each node in the original $G_1'$. The graph $G_1^r$ will now have a clique of size $k_1 + r$ iff $G_1$ has a clique of size $k_1$, and moreover no clique of $G_1^r$ can be bigger than $k_1 + r$. That is, $G_1$ has a clique of size $k_1$, iff the maximum-sized clique in $G_1^r$ has size $k_1 + r$.

We now define the graph $G_2^r$ (i) first in a similar way to $G_1^r$, replacing $k_1$ with $k_2$, $N_1$ with $N_2$ and $E_1$ with $E_2$, and then in addition (ii) adding a fresh disjoint instance $K_{k_2+r-1}$.

It will then be the case that $G_r^2$ has a $(k_2 + r)$-clique iff $G_2$ has a $k_2$-clique. Moreover, it is certain that $g_2^r$ has a $k_2 + r - 1$ sized clique and that it has no clique larger than $k_2 + r$. Thus the maximal clique of $G_r^2$ has size $k_2 + r$ iff $G_2$ has a $k_2$-clique.

If $k_1 > k_2$ we can thus take $G' := G_1^0 \times G_2^{k_1-k_2}$, and otherwise $G' := G_1^{k_2-k_1} \times G_2^0$.

**Exercise 4.2**

(a) Assume that **P**=**NP**. Show that then **EXP**=**NEXP**.

*Remark*: Assume that $L$ is decided by some TM running in time $T(n)$ with $T(n)$ time-constructible and $T(n) \in \mathcal{O}(2^{n^c})$ for some $c \geq 1$. Show that then

$$L_{\text{pad}} := \{x10^{T(|x|)}1 \mid x \in L\} \in \mathbf{NP}.$$

*(b) Show that also **EXP**=**NEXP** if only *every unary* **NP**-language is also in **P**.

*Remark*: For $x \in \{0,1\}^*$ let $\langle x\rangle$ be the natural number represented by $x$ assuming lsbf. Given a language $L$ which is decided in time $T(n)$ (with $T(n)$ time-constructable) show that

$$L_{\text{upad}} = \{1^{\langle x10^{|T(n)|}1\rangle} \mid x \in L\} \in \mathbf{NP}$$

with $|T(n)|$ ($\approx \lceil \log T(n) \rceil$) the length of the lsbf representation of $T(n)$.

**Solution:** Let $L \in \mathbf{NEXP}$ be decided the NTM by $N$ in time $T(n) \in \mathcal{O}(2^{n^c})$ for some $c \geq 1$. Further, let $M_T$ be the TM that computes $x \mapsto \text{lsbf}(T(|x|))$ in time $T(|x|)$.

We claim $L_{\text{pad}} \in$ NP: (If a "check" fails, we reject the input.)

- On input $y = 1^m$ first compute $w = \text{lsbf}(m)$.
- Then check that $w = z10^k1$ for some $z \in \{0,1\}^*$ and $k \in \mathbb{N}$.
- Next, simulate $M_T$ on input $z$ for exactly $2^{k+1}$ steps and check that the halting configuration is reached.

  Note that
  $$m = \langle w\rangle = \langle z10^k1\rangle \geq \langle 0^k1\rangle = 2^{k+1}.$$

- As $M_T$ terminates, its output is $\text{lsbf}(T(|z|))$. Check that $k = |T(|z|)|$.
- Now simulate $N$ on $z$ for exactly $2^{k+1}$ steps. As

$$2^{k+1} = 2^{|T(n)|+1} \geq T(n)$$

  the simulation reaches the halting configuration and therefore decides whether $z \in L$ or not.

As we assume that every unary language in **NP** is also in **P**, we also find a TM $M$ which decides $L_{\text{pad}}$ in polynomial time. From $M$ we obtain a TM $M'$ which decides $L$ in **EXP**:

- For input $x$ ($n = |x|$) first compute $\text{lsbf}(T(n))$ in time $T(n)$.
- Then generate $w = x10^{|T(n)|}1$ in time $n + 2 + |T(n)| = n + 2 + \lceil \log T(n) \rceil$.
- Finally, generate $y = 1^{\langle w \rangle}$. Note that

$$|y| = \langle w \rangle = \langle x10^{|T(n)|}1 \rangle \leq \langle 0^{|w|}1 \rangle = 2^{|w|+1} = 2^{2+n+|T(n)|+1} \leq 2^{n+4} \cdot T(n).$$

- Now use $M'$ to decide whether $y \in L_{\text{pad}}$ or not.

## Exercise 4.3

Say that $A$ is *linear-time reducible* to $B$ if there is function $f$ computable in time $\mathcal{O}(n)$ such that $x \in A \Leftrightarrow f(x) \in B$.

- Show that there is no P-complete problem w.r.t. linear-time reductions.

*Hint*: Use the time hierarchy theorem for **DTIME**.

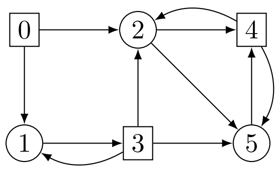
## Exercise 4.4

A *two-person game* consists of a directed graph $G = (V_0, V_1, E)$ (called the *game graph*) whose nodes $V := V_0 \cup V_1$ are partitioned into two sets and a *winning condition*. We assume that every node $v \in V$ has a successor. The two players are called for simplicity player 0 and player 1. A *play* of the two is any finite or infinite path $v_1 v_2 \ldots$ in $G$ where $v_1$ is the starting node. If the play is currently in node $v_i$ and $v_i \in V_0$, then we assume that it is the turn of player 0 to choose $v_{i+1}$ from the successors of $v_i$; if $v_i \in V_1$, player 1 determines the next move. The winning condition defines when a play is won by player 0. E.g.:

- In a *reachability game* the winning condition is simply defined by a subset $T \subseteq V_0 \cup V_1$ (*targets*) of the nodes of $G$, and a play is won by player 0 if it visits $T$ within $n-1$ moves (where $n$ is the total number of nodes of $G$). Hence, player 1 wins a play if he can avoid visiting $T$ for at least $n-1$ moves.
- In a *revisiting game* player 0 wins a play $v_1 v_2 \ldots$ if the first node $v_i$ which is visited a second time belongs to player 0, i.e., $v_i \in V_0$; otherwise player 1 wins the play.

We say that *player $i$ wins node $s$* if he can choose his moves in such a way that he wins any play starting in $s$.

*Example*: Consider the following game graph where nodes of $V_0$ ($V_1$) are of circular (rectangular) shape:



In the reachability game with $T = \{5\}$ player 0 can win node 4: if player 1 moves from 4 to 5, player 0 immediately wins; if player 1 moves from 4 to 2, then player 0 can win again by moving from 2 to 5. On the other hand, player 1 can win node 0 by choosing to always play from 0 to 1 and from 3 to 1.

In the revisiting game played on the same game graph, player 0 can win node 2: he moves from 2 to 5 and then on to 4; no matter how player 1 then chooses to move, the play will end in an already visited node which belongs to player 0. Player 1 can e.g. win node 3 by simply moving to node 1.

(a) Consider a reachability game:

Show that one can decide in time polynomial in $\langle G, s, T\rangle$ if player 0 can win node $s$.

*Hint*: Starting in $T$ compute the set of nodes from which player 0 can always reach $T$ no matter how player 1 chooses his moves.

(b) Consider a revisiting game:

Show that it is **PSPACE**-complete to decide for a given game graph $G$ and node $s$ if player 0 can win $s$.

*Remarks*:

- A game is called *determined* if every node if won by one of the two players.

  Are reachability, resp. revisiting games determined?

- Assume that we change the definition of reachability game by dropping the restriction on the number of moves, i.e., player 0 wins a play if the play eventually reaches a state in $T$.

  Does this change the nodes player 0 can win for a given game graph?

**Solution:**

(a) Let

$$A_0(X) := \{v \in V_0 \mid vE \cap X \neq \emptyset\} \cup \{v \in V_1 \mid vE \subseteq X\}.$$

and

$$W_0 := \bigcup_{k \geq 0} A_0^k(T).$$

Note that $A_0(X)$ can be computed in time $|V||E|$ and $W_0$ in time $|V|^2|E|$ as we can include at most $|V|$ many nodes.

Induction on $k$ shows that player 0 can win any node in $A_0^k(T)$ by simply playing to some node in $A_0^{k-1}(T)$. Any such play has trivially length at most $n-1$ (assuming $T$ is not empty).

Consider any node $v \notin W_0$ and consider any play from $v$ which reaches $T$. There is some smallest $i$ such that $v_i \notin W_0$ and $v_{i+1} \in W_0$. As player 0 can win anny node in $W_0$, we can assume that the remaining play stays in $W_0$. If $v_i$ was in $V_0$, then by definition of $A_0(W)$ we also would have $v_i \in A_0(W_0) = W_0$. So, $v_i \in V_1 \cap W_0$. Hence, player 1 can find a successor of $v_i$ which is not contained in $W_0$, i.e., player 1 can always evade entering $W_0 \supset T$.

$W_0$ is therefore the set of nodes which player 0 can win and $V \setminus W_0$ is the set of nodes which player 1 can win. In particular, reachability games are determined. Note that we didn't really use the restriction

(b) $v$ is won by player 0 iff we do not find a play which is won by player 1. Any play has length at most $n$. So, for a given node $v$, we can enumerate all possible plays in polynomial space and, hence, decided whether $v$ is won by player 0.

In order to see that the revisiting game is **PSPACE**-complete, one simply has to check that the reduction of QBF to the game of geography also works for the revisiting game.

One can also show that the revisiting game is determined: Consider the enlarged game graph, where nodes correspond to plays of length at most $n$. We have an edge from $v_1v_2\ldots v_k$ to $v_1v_2\ldots v_kv_{k+1}$ iff

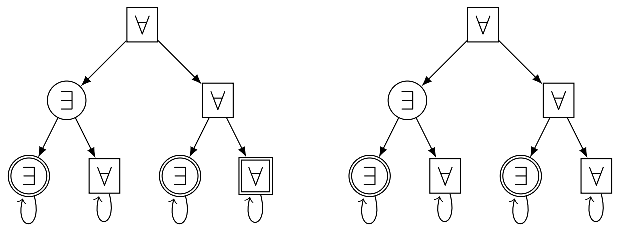
$(v_k, v_{k+1}) \in E$. Set now as target set the sequences which revisit a node of $V_0$ for the first time. Then player 0 wins $v$ in the revisiting game on the original game graph iff he wins $v$ in the reachability game on the enlarged game graph with target set $T$. As the reachability game is determined, the revisiting game is determined too.

**Exercise 4.5**

An *alternating Turing machine* (ATM) $M = (\Gamma, Q_\forall, Q_\exists, \delta_0, \delta_1)$ is an NDTM $(\Gamma, Q_\forall \cup Q_\exists, \delta_0, \delta_1)$ except that (i) the control states are partitioned into sets $Q_\forall$ and $Q_\exists$ and (ii) the acceptance condition is defined as follows:

> Consider the configuration graph $G(M, x)$. We extend the partition of the control states to the configurations (nodes) of $G_{M;x}$: a configuration is in $V_0$ if its control state is in $Q_\exists$; otherwise it is in $V_1$. We then can consider the reachability game played on $G(M, x)$ by the players 0 and 1 where the target set is the set of accepting configurations. $M$ accepts $x$ iff player 0 wins the initial configuration in this reachability game. (For the sake of completeness, assume that every halting/accepting configuration is its unique successor.)

*Example*: Consider the following configuration graphs where accepting configurations have a second circle/rectangle drawn around them. In the left graph the corresponding ATM accepts the input while it rejects the input in the right example:



A language is decided by an ATM $M$ if $M$ accepts every $x \in L$ and rejects any $x \notin L$. The time and space required by an ATM is the time and space required by the underlying NDTM.

The class **AP** consists of all languages $L$ which are decided by an ATM $M$ running in time $T(n) \in \mathcal{O}(n^k)$ for some $k \geq 1$.

(a) An *existential* (*universal*) ATM is an ATM with $Q_\forall = \emptyset$ ($Q_\exists = \emptyset$).

Show that any language $L \in$ **AP** which is decided by an existential (universal) ATM is in **NP** (co**NP**).

(b) Define co**AP** as usual: $L \in$ co**AP** iff $\overline{L} \in$ **AP**.

Show or disprove that **AP** = co**AP**.

(c) Show that QBF is in **AP**.

(d) Show that any $L \in$ **AP** is in **PSPACE**.

*Remark*: Adapt the recursive decision procedure for QBF $\in$ **PSPACE** you have seen in the lecture.

*Remark*: Similarly as **AP**=**PSPACE**, one can show **APSPACE**=**EXPTIME**.

**Solution:**

(a) -

(b) As reachability games are determined, player 0 does not win the initial configuration iff player 1 wins the initial configuration. As the game graph is acyclic, this means that player 1 can force any play from the initial configuration to a rejecting configuration. Swapping 0 and 1 and rejecting and accepting, we therefore obtain an ATM for $\overline{L}$.

(c) The ATM reads the formula from left to right. Its control state is universal iff it reads an universal quantifiers. For every quantifier read, the ATM guesses a truth value. Finally, it simply evaluates the formula on the guessed truth assignment in polynomial time.

(d) If $L \in \mathbf{AP}$, then there must exist a polynomial time bounded Turing Machine $M$ recognising $L$. Say that this time bound is given by $T(n)$ where $n$ is the size of the input. Given a word $w$, $M$ accepts $w$ iff there is an alternating run-tree of $M$ when given input $w$. We may assume w.l.o.g. that all branches of all run-trees of $M$ have length at most $T(|w|)$ (if need be a counter can be added to ensure the machine always terminates after $T(|w|)$ steps).

We can construct a machine $\hat{M}$ that exhaustively searches for such a run-tree using a depth-first search. Note that a configuration of $M$ can be represented as a pair $(q, s)$ (control-state, tape contents) in polynomial space (in polynomially many steps, one can only fill polynomially many cells of the tape).

We can then define a recursive procedure *accepting*$(q, s)$ that returns true iff there exists an accepting-run tree of $M$ from $(q, s)$ (assuming the input $w$) and otherwise returns false. Since paths in a run-tree of $M$ are at most length $T(|w|)$, the call-stack of this recursive procedure will have height at most $T(|w|)$ and so the procedure can run in time $O(T(|w|)^2$.

- If $q$ is accepting, return true, if $(q, s)$ has no successor configuration and $q$ is not accepting, return false.
- If $q$ is an $\exists$-state, return $\delta_0(q, s) \vee \delta_1(q, s)$.
- If $q$ is a $\forall$-state, return $\delta_0(q, s) \wedge \delta_1(q, s)$.

We can then define $\hat{M}$ to be the machine computing *accepting*$(q_0, s_0)$, where $(q_0, s_0)$ is the initial configuration of $M$.