Complexity (SLOa) - Homework 3

Šimon Stupinský - xstupi00@stud.fit.vutbr.cz

Obtained points:

1. TASK (3.5 POINTS)

Assignment: Prove that the membership problem for context sensitive languages $\texttt{MEM}_1$ is **PSPACE**-*complete*.

Context-Sensitive Grammar

A *context-sensitive* grammar $\mathbf{G}$ is a quadruple $(\mathbf{V}, \Sigma, \mathbf{P}, \mathbf{S})$ where:

- $\mathbf{V}$ is finite set of *symbols*, $\Sigma \subseteq \mathbf{V}$ is the finite set of *terminal symbols*, and $\mathbf{S}$ is the *start symbol* in $\mathbf{V} \setminus \Sigma$.
- $\mathbf{P}$ is a finite set of *productions* of the form $\alpha \to \beta$, s.t. $\alpha, \beta \in V^+$ and $|\alpha| \leq |\beta|$.

Language of the grammar. For two words $\mathbf{u}$ and $\mathbf{v}$ in $\mathbf{V}^*$, we say that $\mathbf{u}$ derives $\mathbf{v}$ (denoted $\mathbf{u} \Rightarrow \mathbf{v}$), if there exist $\mathbf{x}, \mathbf{y}, \alpha, \beta \in \mathbf{V}^*$ s.t. $\mathbf{u} = \mathbf{x}\alpha\mathbf{y}$, $\mathbf{v} = \mathbf{x}\beta\mathbf{y}$ and $\alpha \to \beta \in \mathbf{P}$. Let $\overset{*}{\Rightarrow}$ denote the *reflexive* and *transitive* closure of $\Rightarrow$. Using this notation we are ready to describe the *context-sensitive* language defined by the *context-sensitive* grammar $\mathbf{G}$: $\mathbf{L}(\mathbf{G}) = \{\, \mathbf{w} \,|\, \mathbf{S} \overset{*}{\Rightarrow} \mathbf{w} \textbf{ and } \mathbf{w} \in \Sigma^* \,\}$.

PSPACE-Completeness

PTIME. Decision problems solvable in *polynomial* **time**.

PSPACE. Decision problems solvable in *polynomial* **space**.

PSPACE-complete. We define a problem $\mathbf{P}$ to be:

- **PSPACE-hard** if for all $P' \in$ **PSPACE**, $P' \leq P$,
- **PSPACE-complete** if P is **PSPACE**-hard and $P \in$ **PSPACE**.

We note, that $P' \leq P$ denote a *polynomial* reduction from P' to P, thus that there exists **PTIME** Turing Machine computing a reduction function $R : \Sigma^* \to \Sigma^*$ s.t. $w \in P' \iff R(w) \in P$. This means that **PSPACE**-*complete* problems are the *harder* problems of **PSPACE**. To prove, that the given decision problem is **PSPACE**-*complete*, we will need to show that it is in **PSPACE**, and in **PSPACE**-hard.

Membership Problem

Now the *membership problem* $\texttt{MEM}_1$ for a *context-sensitive* language $\mathbf{L}(\mathbf{G})$ is defined as follows:

Input : a word $w \in \Sigma^*$, where Σ is the terminal alphabet of G;
Question : is $w \in L(G)$?

Linear Bounded Automaton

A *linear-bounded automaton* is a type of one-tape, non-deterministic Turing machine acceptor where the input is written between special end-markers and the computation can never leave the space between these markers (nor overwrite them). Therefore, we can say, that a *linear-bounded automaton* is a *Non-deterministic Turing Machine* with space complexity $\mathcal{O}(\mathbf{n})$. For any *context-sensitive language*, there exists a *linear-bounded automaton* which generates it. From this immediately follows that a set of strings is a *context-sensitive* language if, and only if, it is accepted by a *linear-bounded automaton*.

$\texttt{MEM}_1 \in$ *PSPACE*

The input instance of this problem $\texttt{MEM}_1$ includes the *context-sensitive* grammar $\mathbf{G}$ and the word $\mathbf{w} \in \Sigma^*$. From the definition of the *context-sensitive* grammar we know, that the productions cannot decrease the length of the generated word ($|\alpha| \leq |\beta|$). Therefore, we can repeatedly apply the individual productions in reverse, with the eventually reaching of the starting symbol $\mathbf{S}$, as the goal. We need to ensure that the machine always halts and never runs forever in the loop without the achieved of the set goal. Therefore we instantiated the counter which counts the number of performed steps and we terminate the computation after at most $|\mathbf{V}|^{|\mathbf{w}|}$ steps, where $\mathbf{V}$ is finite set of *symbols* of the grammar $\mathbf{G}$. Now we can determine the size of this counter as $|\mathbf{w}| \log |\mathbf{V}|$, which belongs to the $\mathcal{O}(\mathbf{n} \log \mathbf{n})$, where $\mathbf{n}$ represents the length of the given word $\mathbf{w}$ ($\mathbf{n} = |\mathbf{w}|$). This complexity $\mathcal{O}(\mathbf{n} \log \mathbf{n})$ puts the given problem $\texttt{MEM}_1$ to the class $\mathbf{NSPACE}(\mathcal{O}(\mathbf{n} \log \mathbf{n}))$ and with use the *Savitch's theorem* we can conclude that $\mathbf{NSPACE}(\mathcal{O}(\mathbf{n} \log \mathbf{n})) \subseteq \mathbf{NPSPACE} = \mathbf{PSPACE}$.

We can prove this also in the second following way. As we have known, for the given grammar $\mathbf{G}$ we can construct the *linear-bounded automaton* $\mathbf{A_G}$ such, that $\mathbf{L}(\mathbf{A_G}) = \mathbf{L}(\mathbf{G})$. This means, that the constructed *linear-bounded automaton* $\mathbf{A_G}$ accepts the given word $\mathbf{w}$ if, and only if, it is generated by the grammar $\mathbf{G}$: $\mathbf{w} \in \mathbf{L}(\mathbf{A_G}) \iff \mathbf{w} \in \mathbf{L}(\mathbf{G})$. The *linear-bounded automaton* $\mathbf{A_G}$ on input $\mathbf{w}$ may traverse at most $|\mathbf{Q}| \cdot |\mathbf{w}| \cdot |\mathbf{V}|^{|\mathbf{w}|}$ different configurations, where $\mathbf{Q}$ represents the set of states of $\mathbf{A_G}$. Therefore, to decide whether $\mathbf{A_G}$ terminates on the input $\mathbf{w}$, we simulate this *linear-bounded automaton* at most for $|\mathbf{Q}| \cdot |\mathbf{w}| \cdot |\mathbf{V}|^{|\mathbf{w}|}$ steps. If the simulation *accepts*, then we also accept and otherwise, we reject. Similar to the first case, to track the number of the steps, we make use of a binary counter that takes at most $\log |\mathbf{G}| + \log |\mathbf{w}| + |\mathbf{w}| \cdot \log |\mathbf{V}|$ cells. It is clear, that it also belongs to the $\mathcal{O}(\mathbf{n} \log \mathbf{n})$, what puts this problem $\texttt{MEM}_1$ to the same class as in the first case, so $\mathbf{NSPACE}(\mathcal{O}(\mathbf{n} \log \mathbf{n}))$. □

$\texttt{MEM}_1 \in$ *PSPACE*-hard

To prove this, it is sufficient to show that any instance of the already known decision problem that is **PSPACE**-hard can be reduced to an instance of this problem in polynomial time. We will show that this problem is **PSPACE**-hard by constructing the reduction from decision problem called *Quantified Boolean Formulas* (QBF), in other word, that the given problem is at least hard as *Quantified Boolean Formulas* problem. This problem is defined as follows, given a *propositional formula* Φ over the *variables* $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ and *quantifiers* $\mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_n} \in \{\forall, \exists\}$ over the set $\{\texttt{true}, \texttt{false}\}$. This problem is query whether the formula $\mathbf{Q_1 x_1 Q_2 x_2 \ldots Q_n x_n} \Phi$ is valid. For clarity, we recall, that this problem is **PSPACE**-complete.

Thus, we will prove that $\mathbf{QBF} \leq \mathbf{MEM_1}$ via a polynomial time reduction. The technique which we use in this reduction is called *padding*. Since $\mathbf{QBF} \in \mathbf{PSPACE}$, there exists a deterministic Turing machine $\mathbf{M}$ that decides **QBF** problem using at most $\mathbf{p}(\mathbf{n})$ space on inputs $\mathbf{w}$ of length $\mathbf{n}$ ($|\mathbf{w}| = \mathbf{n}$), where $\mathbf{p}$ is a *polynomial* function. Now, we construct the reduction function from **QBF** to $\mathbf{MEM}_1$, which maps the given input instance $\mathbf{w} \in \mathbf{QBF}$ to the output instance $\langle \mathbf{M'}, \mathbf{w'} \rangle$ in the following way. The word $\mathbf{w'}$ we obtain by padding the word $\mathbf{w}$ as follows: $\mathbf{w'} = \mathbf{w}\#^{\mathbf{p}(\mathbf{n})}$, where $\mathbf{n}$ represents the length of the word $\mathbf{w}$, thus also $\mathbf{w'} = \mathbf{w}\#^{\mathbf{p}(|\mathbf{w}|)}$. $\mathbf{M'}$ is the Turing machine that first checks that the input is appropriately encoded, in particular, that it is padded with $\mathbf{p}(\mathbf{n})$ #'s, and then it goes to simulate the machine $\mathbf{M}$ on the input $\mathbf{w}$. $\mathbf{M'}$ accepts the input $\mathbf{w'}$, when it will be correctly padded and encoded, and mainly when $\mathbf{M}$ accepts own input string $\mathbf{w}$, otherwise $\mathbf{M'}$ rejects. From this it is clearly, that $\mathbf{w} \in \mathbf{QBF}$ if and only if $\mathbf{w'} \in \mathbf{MEM_1}$:

$$\mathbf{w'} \in \mathbf{MEM_1} \iff \mathbf{M' \text{ accepts } w'} \iff \mathbf{M \text{ accepts } w} \iff \mathbf{w} \in \mathbf{QBF}.$$

It is easy to see that this reduction can be performed in the *polynomial* time. This reduction needs only *logarithmic* space to compute the correct number of #'s for the padding and to compute the encoding. Furthermore, since any *logarithmic* space computation can be simulated in *polynomial* time, this yield yields a polynomial-time reduction. Furthermore, $\mathbf{M'}$'s behaviour on the string $\mathbf{w'}$ will be that of the *linear-bounded automaton* since $\mathbf{M'}$ only uses $\mathbf{p}(\mathbf{n})$ space on $\mathbf{w}$. □

2. TASK (3 POINTS)

Assignment: Let us consider the function $crypt : \{0,1\}^* \to \{0,1\}^*$ such that $crypt \in FP$ (function computable in polynomial time) and there exists $k \in \mathbb{N}$ such that $\forall x \in \{0,1\}^* : |x|^{\frac{1}{k}} \leq |crypt(x)| \leq |x|^k$. Let us define the following language $IMG_{crypt} = \{w \mid \exists v \in \{0,1\}^*, w = crypt(v)\}$. Prove that if $IMG_{crypt} \in NP \setminus UP$ then $crypt$ is not injective.

UP Class. Call a non-deterministic Turing machine *unambiguous* if it has the following property: For any input **x** there is at most **one** accepting computation. **UP** is the class of languages accepted by unambiguous polynomial-time bounded non-deterministic Turing machines. Equivalently, **UP** is the class of languages of the form $\{\mathbf{x} \mid \exists \mathbf{y}\, \mathbf{R}(\mathbf{x},\mathbf{y})\}$, where **R** is polynomial time computable, polynomially balanced, and for each **x**, there is at most one **y** such that $\mathbf{R}(\mathbf{x},\mathbf{y})$.

Proof. We will lead the proof by the contradiction and therefore, we suppose that the function *crypt* **is** *injective*. Then there is an unambiguous polynomial-time bounded non-deterministic Turing machine **M** that accepts given language $\mathbf{IMG_{crypt}}$ ($\mathbf{L(M)} = \mathbf{IMG_{crypt}}$). **M** on the input $\mathbf{w} \in \{\mathbf{0},\mathbf{1}\}^*$ non-deterministically guesses a string $\mathbf{v} \in \{\mathbf{0},\mathbf{1}\}^*$ of length at least $|\mathbf{w}|^{\frac{\mathbf{1}}{\mathbf{k}}}$ and at most $|\mathbf{w}|^{\mathbf{k}}$, recalling the definition of the *crypt* function in the assignment, and tests whether $\mathbf{w} = \mathbf{crypt}(\mathbf{v})$. If the answer is `YES` then **M** accept, since *crypt* is *injective* this will happen at most once. It should be clear that this non-deterministic machine decides $\mathbf{IMG_{crypt}}$ in the polynomial time, and is unambiguous. Hence $\mathbf{IMG_{crypt}} \in \mathbf{UP}$, according to the listed definition of this class. However, this is the **contradiction** to $\mathbf{IMG_{crypt}} \in \mathbf{NP} \setminus \mathbf{UP}$ and therefore our assumption, that the function *crypt* **is** *injective* was incorrect. Therefore, we proved, that if $\mathbf{IMG_{crypt}} \in \mathbf{NP} \setminus \mathbf{UP}$ then the function *crypt* **is not** *injective*. □.

---

3. TASK (3.5 POINTS)

Assignment: Prove that `#GRAPH COLOURING` is **#P**-complete.

#P Class

#P. We now define a powerful class of functions called **#P**, pronounced "number **P**", "sharp **P**" or even "pound **P**". Let **Q** be a polynomially balanced, polynomial-time decidable binary relation. The *counting problem* associated with **Q** is the following: Given **x**, how many **y** are there such that $(\mathbf{x},\mathbf{y}) \in \mathbf{Q}$? The output required is an integer in binary, say. **#P** is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

#P-completeness. The **#P-complete** problems are *most difficult* problems in the class **#P**. A *counting problem* **w** is **#P-hard** if $\mathbf{v} \leq \mathbf{w}$ for all problems $\mathbf{v} \in$ **#P**. A *counting problem* **w** is **#P-complete** if it is **#P-hard** and $\mathbf{w} \in$ **#P**.

Counting and parsimonious reductions. To prove **#P-hardness**, we need reductions from already known **#P-complete** problems. It is necessary to observe how counting problems **v** and **w** are related under the process of reduction. The notion of *counting reductions* and *parsimonious reductions* have been introduced for this purpose. Let $\mathbf{w} : \Sigma^* \to \mathscr{P}(?^*)$ and $\mathbf{v} : \Pi^* \to \mathscr{P}(\Delta^*)$ be *counting problems*. A *weakly parsimonious* reduction **w** to **v** consists of a pair *polynomial-time* computable functions $\sigma : \Sigma^* \to \Pi^*$ and $\tau : \Sigma^* \times N \to N$ such that $|w(x)| = \tau(x, |v(\sigma(x))|)$. A *counting reduction* σ is *parsimonious* if $|w(x)| = |v(\sigma(x))|$. If **w** is **#P-complete** and there is a weakly parsimonious reduction from **w** to **v** then **v** is **#P-hard**. The reductions in the **#P-hardness** proofs must preserve the number of solutions, possibly with a certain factor.

Graph Colouring

A *Graph k-colourability* is an assignment of colours $\{\mathbf{1},\mathbf{2},\ldots,\mathbf{k}\}$ to the vertices of graph **G** in such way that neighbour vertices of graph should receive different colours. That means, in a proper graph colouring, if two vertices **u** and **v** of a graph share an edge $(\mathbf{u},\mathbf{v})$, then they must be coloured with different colours. Graph colouring was among the 21 **NP**-complete problems originally given by Richard Karp in the year 1972. The *counting problem* is called *chromatic polynomial* and it counts the number of ways a graph can be coloured using no more than a given number of colours. It is a function $\mathbf{P}(\mathbf{G},\mathbf{k})$ that counts the number of k-colouring of **G**.

#GRAPH-COLOURING $\in$ #P

The colouring is the *certificate*, i.e., a list of vertices and their colours, then we can to construct the following *verifier* **V** for this problem. **V** on the input $< \mathbf{G},\mathbf{c} >$ firstly checks whether **c** includes $\leq \mathbf{k}$ colours. The individual colour of each vertex of **G** is specified by **c**. Subsequently, **V** for each vertex checks whether it has a unique colour from each of its neighbours. If all check pass, then **V** accept, otherwise, reject. In summary, we have given a graph **G**, and a colouring assignment of the vertices, then we can simply walk the graph and make certain that all adjacent vertices have a different colour, and make certain that only **k** colours are used. This is clearly by $\mathscr{O}(|\mathbf{V}| + |\mathbf{E}|)$, where $|\mathbf{V}|$ is the number of vertices and $|\mathbf{E}|$ is the number of edges of graph **G**.

#GRAPH-COLOURING $\in$ #P-hard

Firstly, we show the part that maps instances of **#3SAT** to relevant instances of **#3COLOURING**. For a better explanation, we reduce the instance of **#3SAT** to the instance of **#3COLOURING** firstly, and then reduce this instance of **#3COLOURING** to the instance of **#4COLOURING** problem. Then, in fact, we can reduce **#kCOLOURING** to **#k+1COLOURING**, for $\mathbf{n} \geq \mathbf{3}$, by the same way as we will prove. Now, we generalised the reduction approach to reduce any instance of **#3SAT** formula to a **#kCOLOURING** graph in polynomial time with mathematical proof.

3SAT $\leq$ 3COLOURING. To show that **3COLOURING** is **#P**-hard, we reduce from **3SAT**. Given an instance ϕ of **3SAT**, we construct the reduction that will produce the following graph **G**. The graph has three special vertices **A, B, C** and one vertex for each literal, $\mathbf{x_i}$ and $\overline{\mathbf{x_i}}$, then for each such literal, we construct a triangle $\forall \mathbf{i} : \mathbf{A}, \mathbf{x_i}, \overline{\mathbf{x_i}}$. Notice that any **3COLOURING** of the graph so far must assign distinct colours to the vertices **B** and **C**, which we mark suggestively as **0** and **1**. Also each pair $\mathbf{x_i}$ and $\overline{\mathbf{x_i}}$ must be coloured with **0** and **1**, or **1** and **0**. Therefore, we can think of the colouring of the *literal vertices* as a truth (logical) assignment.

For each clause $\mathbf{C_J} = (\mathbf{x} \vee \mathbf{y} \vee \mathbf{z})$ appearing in ϕ we create a small *OR-Gadget Graph* and it connect to vertices corresponding to $\mathbf{x},\mathbf{y},\mathbf{z}$. *OR-Gadget Graph* has the following properties: if $\mathbf{x},\mathbf{y},\mathbf{z}$ are coloured **0** in a **3COLOURING** then output vertex of OR-gadget has to be coloured **0**; if one of $\mathbf{x},\mathbf{y},\mathbf{z}$ is coloured **1** then OR-gadget can be 3-coloured such that output vertex is coloured **1**. We connect the output vertex of each OR-Gadget to both vertices **A** and **C**. It is clear, that this reduction runs in polynomial time.

Firstly, we start with a `YES` instance of **3SAT**, then we claim that the reduction produces also a `YES` instance of **3COLOURING**. Consider a satisfying assignment for input instance ϕ. We colour the vertices **A**, **B** and **C** with the colours **2**, **0** and **1**, respectively. If $\mathbf{x_i}$ is `true` in the satisfying assignment, then we colour $\mathbf{x_i}$ and $\overline{\mathbf{x_i}}$ with colours **0** and **1**. If $\mathbf{x_i}$ is `false` in the satisfying assignment, then we colour $\mathbf{x_i}$ and $\overline{\mathbf{x_i}}$ with colours **1** and **0**. Now notice that every one of the OR-gadget has among its inputs at least one vertex that is coloured **0**, since every clause has at least **one** true literal in the satisfying assignment. Moreover, the output vertex of each OR-gadget is coloured to **0**, because of its above-described property. Thus, by this observation, we can extend the **3COLOURING** to a **3COLOURING** of the clause OR-Gadgets, obtaining a **3COLOURING** of the entire graph **G**.

Now, if **G** is a `YES` instance of **3COLOURING**, then we claim that the reduction started with a satisfiable formula ϕ. Suppose that we have a **3COLOURING** of **G**, then vertices **A**, **B** and **C** must be coloured with three distinct colours, and let us the colour assigned to **B** as **0** and the colour assigned to **C** as **1**. As we already know, each pair of vertices $\mathbf{x_i}$ and $\overline{\mathbf{x_i}}$ must be coloured with **0** and **1**, or with **1** and **0**, respectively. The output vertex of each clause OR-gadget is coloured with **0**, and so by its properties, the only way it can be **3-coloured** is if at least of its input vertices are coloured with **0**. But this means that we can set $\mathbf{x_i}$ to `true` if its colour is **0** and to `false` if its colour is **1**, and the resulting assignment have to satisfy every clause. Thus the formula ϕ is satisfiable, as was required at the beginning.

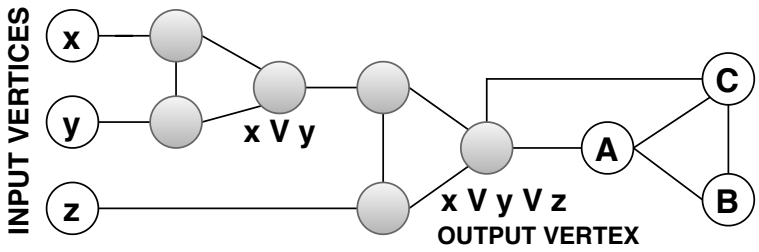

Figure 1. The structure of the *OR-Gadget-Graph*, which is created for each clause $x \vee y \vee z$.

3SAT $\leq$ kCOLOURING. To prove this, we construct the reduction from **k-1COLOURING** to **kCOLOURING**, or show that **k** $-$ **1COLOURING** $\leq$ **kCOLOURING**. Let $\mathbf{G_{k-1}}$ be an instance of **k-1COLOURING**, we construct a new graph $\mathbf{G_k}$ as follows: Add a single extra vertex $\mathbf{B_{k-3}}$ and connect it to every other vertex in the graph $\mathbf{G_{k-1}}$. This is clearly polynomial in the size of the graph. Now we must show that $\mathbf{G_k}$ is a `YES` instance of **kCOLOURING** if and only if $\mathbf{G_{k-1}}$ is a `YES` instance of **k-1COLOURING**.

Assume that $\mathbf{G_{k-1}}$ is (**k-1**)-colourable. Therefore, $\mathbf{G_k}$ is **k**-colourable because the added vertex $\mathbf{B_{k-3}}$, which is connected to all the other vertices in the graph, can be coloured with a $\mathbf{k^{th}}$ colour, and it will always be connected to vertices that are **one** of **k-1** other colours. In the second direction, we assume that $\mathbf{G_k}$ is **k**-colourable. Because the vertex $\mathbf{B_{k-3}}$ is connected to every vertex in the graph, it must be the only vertex in the graph $\mathbf{G_k}$ that has a certain colour. Therefore, all other vertices in the graph $\mathbf{G_k}$ are coloured **one** of **k-1** colours and thus, the graph $\mathbf{G_{k-1}}$ is (**k-1**)-colourable.

Morally parsimonious reduction. To complete the proof that **#kCOLOURING** is **#P-hard**, respectively **#P-complete**, we need to show the recovering between the number of solutions between reduced problems, respectively their instances. One should be careful when dealing with *parsimonious reductions*, since not all **NP**-reductions are parsimonious. We show that the constructed reduction **#3SAT** $\leq$ **#kCOLOURING** is not parsimonious. The intuition here is that, for an instance of **#3SAT**, which is a boolean function ϕ, then ϕ is reduced to a graph **G**, as we have shown above. We already have known, that the ϕ is satisfiable if and only if **G** is **k**-colourable and we could to convert the graph **G** to an assignment for ϕ. For instance, we assume that the vertices sets $\mathbf{V_0}$, $\mathbf{V_1}$ and $\mathbf{V_2}$ are coloured with **0**, **1** and **2** accordingly. The we totally change all the vertices in $\mathbf{V_0}$ to be coloured with **1**, and all the vertices in $\mathbf{V_1}$ to be coloured with **0**. This swap of the colours produces another valid solution but the corresponding boolean assignment remains the same. There are in totally **6** such permutations of colours for presented instance of **#3COLOURING**, therefore an assignment of ϕ would correspond to **6** different colouring of the graph **G**.

When we generalise this observation to the instances of **#kCOLOURING** problem, then we will have the sets of the vertices $\mathbf{V_0}, \mathbf{V_1}, \ldots, \mathbf{V_{k-1}}$ (coloured with **0**, **1**, ..., **k-1**) and there are in totally **k!** permutations of colours for relevant instance of graph **G**, therefore **one** assignment of ϕ would correspond to **k!** different colouring. As we defined above, **#A** reduces to **#B**, if there is a pair of polynomial-time computable functions (σ, τ), such that: $\forall x \in \#A : |\#A(x)| = \tau(\#B(\sigma(x)))$. According to this definition and presented features, we can conclude, that the **#3COLOURING** problem is **#P-complete**. □