

Exercises for the lecture

Foundations of Complexity Theory

Solutions to Exercise Sheet 2

for the exercise on October 28, 2026

Exercise 2.1

Answer the following questions. Justify your answers.

- (a) Is the sum of two time- resp. space-constructible functions again time- resp. space-constructible?
- (b) Is the composition of two time- resp. space-constructible functions again time- resp. space-constructible?

Solution: Let M and M' be the corresponding TM for f and f' . It is clear that both TM need to read the input once fully (otherwise $f(n) \geq n$ resp. $f(n) \geq \log n$ cannot be guaranteed).

- (a) Yes (twice). Let the input be a^n . We first simulate M and during the simulation copy the complete input $\vdash a^n \dashv$ to a new tape by simply following the moves on the input tape (and writing the input symbol if it is not present on the extra tape). Clearly, this modification does not introduce additional steps. We keep the head on the right border of the copied input (incl. delimiter) once we reach this symbol (i.e., do not move it anymore in the same manner as the head on the input tape). The TM M terminates in $f(n+2)$ steps and the head on the extra tape rests on the final symbol (delimiter) of a^n . Next, we simulate the second TM M' with the extra tape as input (and otherwise different tapes from the simulation of M) by inverting all moves on the extra tape and the meaning of the delimiters $\vdash$ and $\dashv$. Started on the end delimiter $\dashv$, the simulation ends after $f'(n+2)$ steps and thus the total computation takes $f(n+2) + f'(n+2) = (f + f')(n+2)$ steps.

For space-constructibility we first simulate M with output $a^{f(n+2)}$ in space $f(n+2)$. Then we return to the first position in the input and start the simulation of M' with additional output $a^{f'(n+2)}$ in space $f'(n+2)$ on additional tapes. Overall we generate the output $a^{f(n+2)+f'(n+2)}$ in space $f(n+2) + f'(n+2) = (f + f')(n+2)$.

- (b) Yes for time-constructibility. Note that we use a special composition to reflect the proper accounting of the delimiters, so we let $(f; f')(n) = f'(f(n) + 2)$ for all n (as we do not require outputs to be produced with delimiters). We adjust M such that it outputs symbol ' a ' on an additional tape in each step. We simulate TM M' on this extra tape

as input. First we simulate the steps of M' on the left delimiter $\vdash$ (and write it before leaving the cell). As soon as we move right, we start the standard simulation. If the head arrives on a blank cell after a step to the right, then we simulate another step of M , which will generate another ' a ' on this tape, and at the same time (part of the same step) we simulate a step of M' on this not yet written ' a '. On all other symbols we simulate M' directly. If M terminates during the simulation of a step, then we write the right delimiter $\dashv$ onto the extra tape and from then on only simulate M' on the now completely written input. In total we will generate $\vdash a^{f(n+2)} \dashv$ on the extra tape, but each step of M occurs as part of a step of M' . Since M' runs on the input $\vdash a^{f(n+2)} \dashv$, the simulation needs exactly $f'(f(n+2) + 2) = (f; f')(n+2)$ steps.

In general, composition is not space-constructible. Let $f(n) = n$ and $f'(n) = \log n$ for all n . Then $(f; f')(n) = \log(n+2)$ for all n . According to space hierarchy there exist languages L in $\text{DSPACE}(x)$, but not in $\text{DSPACE}(\log x + 2)$ because

$$\lim_{x \rightarrow \infty} \frac{\log(x+2)}{x} = \lim_{x \rightarrow \infty} \frac{1}{(x+2) \cdot \ln 2} = (\ln 2)^{-1} \cdot \lim_{x \rightarrow \infty} \frac{1}{x+2} = 0 .$$

(Additionally function $(f'; f')$ does not fulfill $(f'; f')(n) \geq \log n$.)

However, if $f'(n) \geq n$ for all n , then $f; f'$ is space-constructible. For this space-constructibility we first simulate M on input a^n with its output on an extra tape and then simulate M' with the extra tape as input. We first write $\vdash$ on this extra tape and then proceed normally. Once M stops, we write a final $\dashv$ on the extra tape and walk back to $\vdash$ on this tape. Thus M needs space $f(n+2)$ by assumption and due to output on the extra tape of length $f(n+2) + 2$ the space consumption is $f(n+2) + 2$. Since $f'(f(n+2) + 2) \geq f(n+2) + 2$ this space is sufficient.

The simulation of M' starts on the extra tape $\vdash a^{f(n+2)} \dashv$ and runs normally (on additional tapes from the previous tapes) and thus utilizes only $f'(f(n+2) + 2)$ space with output $a^{f'(f(n+2)+2)}$.

Exercise 2.2

Show that the following variants VC-C and VC-O of vertex cover are efficiently solvable provided that the classical problem 'vertex cover' VC is efficiently solvable.

- VC** Input: Finite undirected graph (E, K) and $k \in \mathbb{N}$
Output: Does there exist vertex cover E' with $|E'| \leq k$?
($E' \subseteq E$ is vertex cover if $\{e, e'\} \cap E' \neq \emptyset$ for all edges $(e, e') \in K$)
- VC-C** Input: Finite undirected graph (E, K) and $k \in \mathbb{N}$
Output: Vertex cover E' with $|E'| \leq k$ if such exists
- VC-O** Input: Finite undirected graph (E, K)
Output: Vertex cover of minimal cardinality

Solution: Let A be polynomial-time algorithm for VC. Obviously an efficient solution for VC-O also yields efficient solution for VC-C because if $|\hat{E}| \leq k$ for the vertex cover $\hat{E}$ of minimal cardinality, then we return $\hat{E}$ and otherwise no such cover exists. To solve VC-O we first compute the minimal cardinality of a vertex cover via binary search (as in the lecture) with the initial interval $[0, |E|]$. Clearly, no more than $|E|$ vertices are needed for a vertex cover. In this manner we can efficiently determine the optimal cardinality $\hat{k}$ (with a number of calls to A that is logarithmic in $|E| + 1$).

(a) Let $E' = \emptyset$ and $K' = K$

(b) For all $e \in E$ in some order

- If $A((E, K' \cup \{(e, e)\}), \hat{k})$, then set $E' \leftarrow E' \cup \{e\}$ and $K' \leftarrow K' \cup \{(e, e)\}$

In other words, once we identify a vertex e that is part of a suitable vertex cover, then we force it into any vertex cover by adding (and keeping) the self-loop (e, e) . This makes the correctness proof slightly simpler at the expense of some (minimal) overhead (as well call A on graphs with more and more edges). Obviously, the number of calls to A is linearly bound by $|E|$ and the graph grows at most by $|E|$ edges.

Correctness: Let E' and K' be the values of the variables at termination of the algorithm. Since $E' \neq \emptyset$ (only possible for trivial graph without edges), the last successful call of A was $A((E, K'), \hat{k})$. Additionally, $(e, e) \in K'$ for all $e \in E'$. Since the call to A on (E, K') was successful, there exists a vertex cover E'' of minimal size $\hat{k}$ such that $E' \subseteq E''$ (the self-loops on the vertices of E' force the vertices of E' into the vertex cover). For the converse, let $e \in E''$. Then e was considered inside the loop and the call $A((E, K'' \cup \{(e, e)\}), \hat{k})$ for some $K \subseteq K'' \subseteq K'$ is necessarily successful because E'' is a vertex cover for (E, K') of size $\hat{k}$ and $e \in E''$, which yields that E'' is also a vertex cover for $(E, K'' \cup \{(e, e)\})$. The successful call yields $e \in E'$ as desired.

In the exercise we discussed a variant in which a suitable vertex e is identified via the addition of a self-loop and then removed together with its adjacent edges from the graph (as these edges are covered by taking e into the vertex cover). This approach is also suitable since it also fixes e to be in the vertex cover (which in the approach shown earlier is achieved by keeping the self-loop in the graph), removes the obligations for edges already covered and requires the remaining graph to be covered by a vertex cover of size decreased by 1. The main statement for this approach is: A vertex cover of size k exists for the original graph, if and only if a vertex cover of size $k - 1$ for the remaining graph exists.