ETH zürich
Departement Informatik

Theoretische Informatik

Prof. Dr. Juraj Hromkovič
Dr. Hans-Joachim Böckenhauer
https://courses.ite.inf.ethz.ch/theo_inf_21

Exemplary Solutions – Sheet 8

Zürich, November 26, 2021

Solution to Exercise 22

(a) To prove $L_{\mathrm{H}}^{\complement} \leq_{\mathrm{EE}} L_{\mathrm{all}}$, we describe an algorithm F that transforms an input x for $L_{\mathrm{H}}^{\complement}$ into an input $f(x)$ for L_{all}. The algorithm F first checks if x has the form $\mathrm{Kod}(M)\#w$, for a Turing machine M and a word $w \in \{0,1\}^*$. If this is not the case, then F outputs the code of a Turing machine M_{all} that accepts every input, i.e., $f(x) = \mathrm{Kod}(M_{\mathrm{all}})$. Otherwise, F computes the code of a Turing machine M' that works as follows. On an input y, M' simulates $|y|$ steps of the machine M on w. (If M halts in less steps during the simulation, then M' simulates less than $|y|$ steps.) If M halts on w during the simulation of at most $|y|$ steps, then M' rejects the input y. Otherwise, M' accepts the input y.

The output of F is $f(x) = \mathrm{Kod}(M')$.

Now we prove the correctness of the reduction, i.e., $x \in L_{\mathrm{H}}^{\complement} \iff f(x) \in L_{\mathrm{all}}$, for all $x \in \{0,1,\#\}^*$.

Let us first suppose that $x \in L_{\mathrm{H}}^{\complement}$ and distinguish the following two cases. If x does not have the form $\mathrm{Kod}(M)\#w$, for any Turing machine M and any word $w \in \{0,1\}^*$, then $f(x) = \mathrm{Kod}(M_{\mathrm{all}})$ and thus $f(x) \in L_{\mathrm{all}}$. Otherwise, if $x = \mathrm{Kod}(M)\#w$, for a Turing machine M and a word $w \in \{0,1\}^*$, then M does not halt on w because $x \in L_{\mathrm{H}}^{\complement}$. This further implies that M' accepts every input $y \in \Sigma^*$ because M never halts on w during the simulation, in particular not within $|y|$ steps. Hence, $f(x) \in L_{\mathrm{all}}$ holds.

Let us suppose that $x \notin L_{\mathrm{H}}^{\complement}$, i.e., $x \in L_{\mathrm{H}}$. Then x has the form $\mathrm{Kod}(M)\#w$, for a Turing machine M and a word $w \in \{0,1\}^*$, and M halts on w after a certain number of steps k. Let y be an arbitrary word in Σ^k. On the input y, M' simulates $|y| = k$ steps of M on w. Because M halts on w during $|y| = k$ steps, M' rejects the input y, i.e., $y \notin L(M')$. Hence, $f(x) = \mathrm{Kod}(M') \notin L_{\mathrm{all}}$.

(b) We have

$$(L_{\text{infinite}})^{\complement} = \{w \in \{0,1\}^* \mid w \neq \text{Kod}(M') \text{ for all TM } M'\} \\ \cup \{\text{Kod}(M) \mid \text{there exists some input } x \text{ on which } M \text{ halts}\}.$$

In the following, we describe a nondeterministic Turing machine N with $L(N) = (L_{\text{infinite}})^{\complement}$. Hence, for every word $w \in (L_{\text{infinite}})^{\complement}$, there exists a finite accepting computation of N on w. The NTM N first checks if the input w has the form $w = \text{Kod}(M)$, for some TM M. If this is not the case, then N accepts the input w. If $w = \text{Kod}(M)$ holds for some TM M, then N nondeterministically chooses a word x over the input alphabet of M and simulates M on x deterministically.

If M halts on x, then N accepts its input w. If M does not halt on x, then N does not halt on w either.

The machine N clearly accepts all words that are not codes of a Turing machine. If the input w is the code of a TM M, then $w \in (L_{\text{infinite}})^{\complement}$ holds if and only if there exists a word x such that M halts on x. Hence, if $w \in (L_{\text{infinite}})^{\complement}$, then there exists an accepting computation of N on w in which N nondeterministically chooses exactly this word x. If $w \in L_{\text{infinite}}$, then no such accepting computation of N on w exists. Hence, N is a NTM that accepts $(L_{\text{infinite}})^{\complement}$ and thus $(L_{\text{infinite}})^{\complement} \in \mathcal{L}_{\text{RE}}$.

Solution to Exercise 23

The idea behind the construction of A is to simulate 12 steps of M in 6 steps of A. To this end, every 12 cells of the working tape of M are combined into one cell of A. The same compression is applied to the input as well, A uses its second working tape for that. We note that A can simulate a constant number of M's computation steps in a single step if A has saved the symbols read by M in those steps in its state in advance. We thus pay for optimizing the running time by a significant blow-up of the working alphabet and the set of states.

The MTM A has the same input on its input tape as M. To shorten the computation, A first compresses this input to the second working tape. To this end, it always reads 12 cells of the input tape and writes the 12-tuple of input symbols in one cell of the second working tape. This clearly requires $n + 1$ steps on an input of length n since the head on the input tape reaches the right endmarker \$ at the $(n + 1)$-th step. Afterwards, A moves the head on the second working tape back to the start. Because $\lceil n/12 \rceil$ cells are used on that tape, this requires $\lceil n/12 \rceil$ steps. Hence, A needs $\frac{13n}{12} + c_1$ steps in total for the preprocessing, for a small constant c_1, e.g., $c_1 = 2$.

The actual simulation of M by A proceeds in *rounds* of up to 6 computation steps. In every round, A simulates 12 computation steps of M. This yields a running time of the simulation at most

$$\left\lceil \frac{\text{Time}_M(n)}{12} \right\rceil \cdot 6 \leq \frac{\text{Time}_M(n)}{2} + c_2\,,$$

for some small constant c_2, e.g., $c_2 = 6$.

At the beginning of every round, A reads the contents of its two working tapes at the current cell and the two neighbouring cells at the left and right and saves these $3 \cdot 12 = 36$ cells of the input and working tape of M in its states. This requires 4 steps: one step to the left, two steps to the right, and one step back to the original position. Hence, A now has enough information to simulate 12 steps of M in its states since M can move by at most 12 positions in 12 steps and A knows at least 12 cells to the left and right of the current position of M. In the 12 simulated steps, M can only change the contents of two of the three blocks of 12 cells. These changes can be performed by A in at most 2 steps on its tapes as follows: We only describe the modification of the first working tape, changing the head position on the simulated input tape can be performed analogously.

In the fifth step of the round, A changes the content of the current cell of the working tape according to the 12 computation steps of M and moves the head to the left or right if changes are required in the corresponding part of M's tape simulated there. In the sixth step, A performs modifications on the neighbouring cell and potentially returns back. If changes are only necessary on the current position of both working tapes of A, then the sixth step is not needed.

Overall, this yields the claimed upper bound on the total running time of A.

Solution to Exercise 24

The grammar $G = (\{S, A, X\}, \{0, 1, 2\}, P, S)$ with

$$P = \{S \to AS2, S \to X, AX \to 0X1, A0 \to 0A, X \to \lambda\}$$

generates the language L. It is based on the following idea: Using the rule $S \to AS2$, an equal number of A's and 2's is generated. The A's serve as placeholders for 0's and 1's here. Using the rule $AX \to 0X1$, a symbol A is transformed into a 0 and 1. The rule $A0 \to 0A$ moves the produced 0 across all A's to the left. Once the produced 0 has been moved left at least once, the rule $AX \to 0X1$ can be applied again. If the rule $X \to \lambda$ is applied although A's are still present, then no terminal word can be produced, which means that we do not get a valid derivation. The empty word can also be derived using $S \Rightarrow X \Rightarrow \lambda$. A derivation of the word 000111222 can be as follows:

$$\begin{aligned} S &\Rightarrow AS2 \Rightarrow AAS22 \Rightarrow AAAS222 \Rightarrow AAAX222 \\ &\Rightarrow AA0X1222 \Rightarrow A0AX1222 \Rightarrow 0AAX1222 \Rightarrow 0A0X11222 \\ &\Rightarrow 00AX11222 \Rightarrow 000X111222 \Rightarrow 000111222\,. \end{aligned}$$