ETH zürich
Departement Informatik

**Theoretische Informatik**

Prof. Dr. Juraj Hromkovič
Dr. Hans-Joachim Böckenhauer
`https://courses.ite.inf.ethz.ch/theoInf20`

# Solution Suggestions – Sheet 1

Zürich, September 25, 2020

## Solution to Exercise 1

(a) First, we count how many words of length $n$ over $\{a, b, c\}$ do *not* satisfy the condition from the exercise. These are all those words that contain at most two of the letters. There are $2^n$ different words that contain only the letters $a$ and $b$ because one of the two letters can be chosen for each for each of the $n$ positions. Analogously, there are $2^n$ different words containing only $a$ and $c$ and $2^n$ different words containing only $b$ and $c$. Among these $3 \cdot 2^n$ words we have counted the words $a^n$, $b^n$, and $c^n$ twice, thus there are in total $3 \cdot 2^n - 3$ words that do not satisfy the condition from the the exercise. Since there are in total $3^n$ words of length $n$ over $\{a, b, c\}$, we have in total

$$3^n - 3 \cdot 2^n + 3$$

words containing each of the three letters at least once.

(b) Let $\Sigma = \{0, 1\}$ and $n \in \mathbb{N}$. We denote by $L_n$ the set of words from $\Sigma^n$ not containing the subword 11 and set $N(n) = |L_n|$. We want to determine $N(n)$ recursively. Evidently, we have $N(0) = 1$ since there is only the word $\lambda$ of length 0 over $\{0, 1\}$ and it does not contain 11 as a subword. Moreover, we have $N(1) = 2$ since the words 0 and 1 both do not contain 11 as a subword.

If we consider a word $w \in L_{n+1}$ with $n \geq 1$, then we can write $w$ as $w = xab$ with $a, b \in \{0, 1\}$ and $x \in L_{n-1}$. If $b = 0$, then $xa$ can be an arbitrary word from $L_n$. If $b = 1$, then we must have $a = 0$, but $x$ can be an arbitrary word from $L_{n-1}$. Thus we obtain for $N(n)$ the recurrence equation

$$N(n + 1) = N(n) + N(n - 1)\,.$$

This is evidently the known recurrence of the Fibonacci numbers, which are defined by $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n - 1) + F(n - 2)$ for all $n \geq 2$. Because of the initial conditions $N(0) = 1$ and $N(1) = 2$ we have

$$N(n) = F(n + 2)\,.$$

The sequence of Fibonacci numbers can be described explicitly by the Binet's formula as

$$F(n) = \frac{\varphi^n - (1 - \varphi)^n}{\sqrt{5}}\,,$$

where $\varphi = \frac{1+\sqrt{5}}{2}$. Thus we obtain for $N(n)$ the explicit representation

$$N(n) = \frac{\varphi^{n+2} - (1-\varphi)^{n+2}}{\sqrt{5}} .$$

## Solution to Exercise 2

(a) The statement is false; there is no such language. Suppose there were a non-empty finite language $L \neq \{\lambda\}$ such that $L^2 = L$. Then there would be a word $w \in L$ of maximal length such that $|w| = \max\{|v| \mid v \in L\}$. From $L \neq \{\lambda\}$ follows $|w| \geq 1$. By the assumption $L^2 = L$ we have $w^2 \in L$. This is a contradiction to the assumption that $w$ is a word of maximal length in $L$ because $|w^2| > |w|$.

(b) The statement is correct. We choose

$$\begin{aligned} L_1 &= \{\lambda, 0\}, \\ L_2 &= \{0^{2i} \mid i \in \mathbb{N}\}, \\ L_3 &= \{0^{2i+1} \mid i \in \mathbb{N}\} \cup \{\lambda\}. \end{aligned}$$

Then we have $L_2 \cap L_3 = \{\lambda\}$ and thus

$$L_1 \cdot (L_2 \cap L_3) = \{\lambda, 0\}$$

is finite. Evidently we also have

$$L_1 \cdot L_2 = \{0^i \mid i \in \mathbb{N}\} \text{ und } L_1 \cdot L_3 = \{0^i \mid i \in \mathbb{N}\}$$

and thus

$$L_1 \cdot L_2 \,\cap\, L_1 \cdot L_3 = \{0^i \mid i \in \mathbb{N}\} = \{0\}^* = \Sigma^* .$$

## Solution to Exercise 3

We show the two directions of the claim separately. First, let $L$ be an infinite recursive language over some alphabet $\Sigma$. Then there is an algorithm $A_r$ recognizing $L$, that is, computing for each word $x \in \Sigma^*$ in finite time the output $A_r(x)$ with $A_r(x) = 1$ if $x \in L$ and $A_r(x) = 0$ if $x \notin L$. We can now use $A_r$ to design an algorithm $A_z$ enumerating $L$. On input $n \in \mathbb{N} - \{0\}$, the algorithm $A_z$ simply enumerates in a loop the words from $\Sigma^*$ in canonical order and tests for each word with the help of $A_r$ whether it lies in $L$. If yes, then the word is output and a counter is incremented by 1; otherwise not. As soon as $n$ words have been output, $A_z$ terminates. This way, $A_z$ is evidently enumerating exactly the canonically first $n$ words from $L$.

For the converse, let $A_z$ be an algorithm enumerating an infinite language $L$ over an alphabet $\Sigma$. Then we can use $A_z$ to design an algorithm $A_r$ recognizing $L$. For each given word $x \in \Sigma^*$, algorithm $A_r$ computes the position $n$ of $x$ in the canonical order of all words and then calls the enumerating algorithm $A_z$ on the input $n$. This way, $A_z$ outputs the canonically first $n$ words from $L$, thus also considering in any case at least the canonically first $n$ words from $\Sigma^*$ and thus also $x$. If the word $x$ is contained in $A_z$'s output, then $A_r$ gives the answer 1 because in this case we have $x \in L$. Otherwise, $A_r$ gives the answer 0 because then we have $x \notin L$.