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 2

Zürich, October 8, 2021

Solution to Exercise 4

(a) We define the sequence $(x_n)_{n=1}^{\infty}$ with $x_n = 0^{2^n}$ for every $n \in \mathbb{N} - \{0\}$. It is clear that all words x_n are pairwise distinct.

Now we provide a program that produces x_n, for every $n \in \mathbb{N} - \{0\}$:

```
begin
  s := n;
  j := 1;
  for i := 1 to s do
    j := j * 2;
  for i := 1 to j do
    write(0);
end.
```

This program first computes $j = 2^n$ in a loop. A subsequent loop prints 2^n-times the symbol 0, which yields exactly the word x_n.

The only part of the source code that depends on x_n is the represenation of n in the second line. The rest of the source code is of constant length. Hence, the binary length of the source code is at most $\lceil \log_2(n+1) \rceil + c$, for a constant c.

This yields the upper bound on the Kolmogorov complexity of x_n

$$K(x_n) \leq \lceil \log_2(n+1) \rceil + c \leq \log_2 n + (c+1)$$

for a constant c.

The length of x_n is $|x_n| = |0^{2^n}| = 2^n$. Hence, $\log_2 \log_2 |x_n| = \log_2 n$ and we derive the upper bound

$$K(x_n) \leq \log_2 \log_2 |x_n| + (c+1)$$

on the Kolmogorov complexity of x_n.

(b) We present an indirect proof that no such sequence $(x_n)_{n=1}^{\infty}$ of pairwise distinct words exists. Suppose that $(x_n)_{n=1}^{\infty}$ is such a sequence. Then there exists a constant $c \in \mathbb{N}$ such that

$$K(x_n) \leq \log_2 \sqrt{n} + c$$

holds for all $n \in \mathbb{N} - \{0\}$.

For a fixed $n \in \mathbb{N} - \{0\}$ and $t := \log_2 \sqrt{n} + c$, there are at most

$$\sum_{i=0}^{t} 2^i = 2^{t+1} - 1 = 2^{\log_2 \sqrt{n}+c+1} - 1 = \sqrt{n} \cdot 2^{c+1} - 1$$

source codes of binary length at most t, thus there are at most $\sqrt{n} \cdot 2^{c+1} - 1$ pairwise distinct words that are produced by source codes of binary length at most t. For a sufficiently large n, which only depends on the constant c, we have $\sqrt{n} \cdot 2^{c+1} - 1 < n$ and this is a contradiction to the fact that the n words $x_1, \ldots, x_n$ are pairwise distinct and $K(x_i) \leq t$ holds for all $i \in \{1, \ldots, n\}$, i.e., $x_1, \ldots, x_n$ are produced by source codes of binary length at most t.

Solution to Exercise 5

We present an indirect proof of the statement. Suppose that there are infinitely many random numbers in the set $\{n^2 \mid n \in \mathbb{N}\}$. By definition, this means that there are infinitely many $n \in \mathbb{N}$ such that

$$K(n^2) \geq \lceil \log_2(n^2 + 1) \rceil - 1 \geq 2 \cdot \log_2 n - 1 \tag{1}$$

holds.

Furthermore, for every $n \in \mathbb{N}$, the number n^2 can be produced by a program C_n that contains the binary representation of n, computes the number n^2, and prints it. All parts of this program except for the representation of n are of constant length. Hence, the binary length of C_n is $\lceil \log_2(n+1) \rceil + c \leq \log_2 n + (c+1)$, for some constant c. It follows that

$$K(n^2) \leq \log_2 n + (c+1) \tag{2}$$

holds.

The two bounds (1) and (2) on the Kolmogorov complexity yield

$$2 \cdot \log_2 n - 1 \leq \log_2 n + (c+1)$$

for infinitely many $n \in \mathbb{N}$. This further implies that

$$\log_2 n \leq c + 2$$

must also hold for these infinitely many n, but this is impossible because c is a constant while $\log_2 n$ grows arbitrarily for increasing n. Hence, our assumption is false and the statement to be proved holds.

Solution to Exercise 6

There clearly exists a program A_L that, for a given word $x \in \Sigma^*_{\text{bool}}$, decides whether $x \in L$. By Theorem 2.65, the n-th word x_n from L with respect to the canonical order satisfies

$$K(x_n) \leq \lceil \log_2(n+1) \rceil + c \leq \log_2(n) + (c+1)\,,$$

for some constant c independent of n.

Now we bound n in terms of $|x_n|$ by estimating the number of words that have the form $1^i 0^j 1^k$ and their length $i + j + k$ is at most $|x_n|$. The first n words $x_1, \ldots, x_n$ in L with

respect to the canonical order have the form $1^i0^j1^k$ and their length is $i+j+k \leq |x_n|$. The bound $i+j+k \leq |x_n|$ implies that $i, j, k \leq |x_n|$ and thus $n \leq |x_n|^3$. Finally, it follows that

$$K(x_n) \leq \log_2 n + (c+1) \leq \log_2 |x_n|^3 + (c+1) = 3 \cdot \log_2 |x_n| + (c+1)$$

holds.

Note: One could propose an alternative solution, where a program simply contains binary representations of the values i, j, k and uses them to produce its output. Unfortunately, this is not as easy as it might seem at first glance. Since i, j, and k are allowed to be arbitrary natural numbers, their binary representation might coincide with the representation of other parts of source code (e.g., function names). Hence, we would have to specify where the binary representations of the values i, j, and k are exactly located so that they are properly interpreted. For a single number i, this can be easily done by storing the lengths of the constant parts of the source code before and after the representation of i using only constant space. But for two or more numbers, it is not sufficient to store such constant lengths (before, between, and after the binary representations) to unambiguously interpret the source code. It could happen that there are several possibilities to interpret the values i, j, and k and the surrounding source code.

Hence, one has to use a self-delimiting encoding for the numbers i, j, and k (cf. the encoding in the proof of the prime number theorem in the textbook). But this leads to an increase in the binary length of the source code. A simple self-delimiting encoding (insert a zero after every but the last bit, insert a one after the last bit so that the encoding can be uniquely interpreted) yields a source code of binary length $2 \cdot (\log_2 i + \log_2 j + \log_2 k) + c$, for some constant c, i.e., a source code of binary length up to $6 \cdot \log_2 |x_n| + c$.

To achieve a binary length of $3 \cdot \log_2 |x_n| + c$, the binary representations of i, j, k must have the same length. Then it suffices to store the lengths of the constant parts of the source code before and after the concatenated representation of the three numbers using only constant space, just like in the case of a single number. Afterwards, the numbers i, j, k can be uniquely interpreted by splitting the binary encoding into three parts of equal length. If the numbers i, j, k do not have the same binary length, their binary representations must be padded with leading zeros.