ETH zürich
Departement Informatik

Theoretische Informatik

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

Exemplary Solutions – Sheet 2

Zürich, October 2, 2020

Solution to Exercise 4

(a) We provide, for every $n \in \mathbb{N} - \{0\}$, a Pascal program that prints w_n:

```
begin
  s:= n;
  s:= 3*s*s;
  j:= 1;
  for i:=1 to s do
    j:=j*4;
  for i:=1 to j do
    write(101);
end.
```

This program first computes $s = 3n^2$ and then $j = 4^{3n^2}$ in a loop. In a subsequent loop, the program prints 4^{3n^2} times the string 101, which corresponds exactly to the word w_n.

If we allow the operation ^ denoting exponentiation (which is not the original Pascal syntax, but we want to allow this notation in the following for the sake of readability), the following program prints the word w_n as well:

```
begin
  s:= n;
  s:= 4^(3*s*s);
  for i:=1 to s do
    write(101);
end.
```

The only part of the machine code of both programs depending on w_n is the representation of n in the second line. The remainder of the program code is of constant length. Hence, the binary length of these programs is at most $\lceil \log_2(n+1) \rceil + c$, for some constant c.

This allows us to bound the Kolmogorov complexity of w_n from above by

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

for some constant c.

The length of w_n is $|w_n| = |101| \cdot 4^{3n^2} = 3 \cdot 2^{6n^2}$. It follows that $\log_2 |w_n| = 6n^2 + \log_2 3$ and thus $n = \sqrt{(\log_2 |w_n| - \log_2 3)/6}$. Hence, we get an upper bound of

$$K(w_n) \leq \left\lceil \log_2 \left(1 + \sqrt{(\log_2 |w_n| - \log_2 3)/6} \right) \right\rceil + c \leq \left\lceil \frac{1}{2} \log_2 \log_2 |w_n| \right\rceil + c'$$

on the Kolmogorov complexity of w_n.

(b) We define the sequence $y_1, y_2, y_3, \ldots$ as $y_i = 2^{3^i}$, for all $i \in \mathbb{N}$. Clearly, $y_1 < y_2 < y_3 < \ldots$ and $i = \log_3 \log_2 y_i$, for all $i \geq 1$. The binary representation of y_i is a single 1 followed by 3^i zeros, i. e., $\mathrm{Bin}(y_i) = 1(0)^{3^i}$.

Now we write, for each y_i, a Pascal program C_i that prints $\mathrm{Bin}(y_i)$:

```
begin
  k:= 3^i;
  write(1);
  for j:=1 to k do
    write(0);
end.
```

This program first computes the value of $k = 3^i$, then prints a single 1, and finally k zeros in a `for`-loop. The only part of the machine code of C_i depending on y_i is the representation of i. The remainder of the program code is of constant length. The binary encoding of i is of length $\lceil \log_2(i+1) \rceil \leq 2 + \log_2 i$. Thus,

$$\begin{aligned} K(y_i) &\leq \log_2 i + c \\ &= \log_2 \log_3 \log_2 y_i + c \,, \end{aligned}$$

for some constant c.

Solution to Exercise 5

The interval $[2^n, 2^{n+1} - 1]$ contains 2^n natural numbers of binary length $n+1$. As the binary representations of all these numbers begin with a leading 1, they can be uniquely represented by words of length n over the binary alphabet Σ_{bool}. It thus remains to show that there are at least $2^n - 2^{n-i}$ words w satisfying $K(w) \geq n - i$ among all 2^n words w of length n.

The number of distinct programs of length strictly less than $n - i$ is at most

$$\sum_{j=0}^{n-i-1} 2^j = 2^{n-i} - 1 \,,$$

since there cannot be more machine codes than binary strings, for any given length. Hence, there are at most $2^{n-i} - 1$ words w such that $K(w) < n - i$. The number m of words $w \in (\Sigma_{\mathrm{bool}})^n$ such that $K(w) \geq n - i$ is thus at least

$$m \geq 2^n - (2^{n-i} - 1) > 2^n - 2^{n-i} \,.$$

Solution to Exercise 6

There clearly exists a program A_L that, for a given word $x \in \Sigma_{\mathrm{bool}}^*$, decides whether $x \in L$. By Theorem 2.65, the canonically n-th word x_n from L satisfies

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

for some constants c and c' independent of n. Each word in L is of a length divisible by 3. For each length $3k$, there are $2k+1$ pairwise distinct words in L (one for each possible value of i, for $0 \leq i \leq 2k$). The total number of words in L of length at most $3k$ can thus be computed as

$$\sum_{i=1}^{k}(2i+1) = k + 2\sum_{i=1}^{k} i = k + k \cdot (k+1) = (k+1)^2 - 1\,,$$

for all $k \geq 1$. It follows that every word x_n such that $|x_n| = 3k$ satisfies

$$\begin{aligned} n &\leq (k+1)^2 - 1 \\ &= \frac{|x_n|^2}{9} + \frac{2|x_n|}{3} \\ &\leq |x_n|^2\,. \end{aligned}$$

Note that the last inequality holds for all $k \geq 1$. Hence, every word x_n satisfies

$$\begin{aligned} K(x_n) &\leq \log_2(n) + c \\ &\leq \log_2(|x_n|^2) + c \\ &\leq 2\log_2(|x_n|) + c\,. \end{aligned}$$

Note: One could propose an alternative solution, where a Pascal program simply contains i and j as variable parts and computes the output using these. Unfortunately, this is not as easy as it might seem at first glance. Since i and j are allowed to be arbitrary natural numbers, their binary representation might coincide with the ASCII representation of valid Pascal program fragments. Thus, we would have to tell the compiler, where exactly the values i and j are located within the program, which are not to be interpreted as program text. For a single number i, this can be easily done by denoting the lengths of the constant program parts beore and after the i. This uses only constant additional space (and can, e.g., be encoded as ASCII numbers). But for two numbers, specifying three such lengths is not sufficient for unambiguously decoding the program text. It could happen that there are many possible starting points of the middle part of the program.

Hence, one has to use a self-delimiting encoding for the two numbers i and j (cf. the encoding in the proof of the prime number theorem), But this leads to an increasing length of the machine code. The simplest self-delimiting encoding yields a machine code of length $2(\log i + \log j) + c$, for some constant c.