ETH zürich
Departement Informatik

Theoretische Informatik

Prof. Dr. Juraj Hromkovič
Dr. Hans-Joachim Böckenhauer
http://www.ita.inf.ethz.ch/theoInf18

Lösungsvorschläge – Blatt 2

Zürich, 5. Oktober 2018

Lösung zu Aufgabe 4

Wir zeigen die beiden Richtungen der Behauptung separat.

Sei zunächst L eine unendliche rekursive Sprache über einem Alphabet Σ. Dann existiert ein Algorithmus A_r, der L erkennt, also für jedes Wort $x \in \Sigma^*$ in endlicher Zeit die Ausgabe $A_r(x)$ berechnet mit $A_r(x) = 1$, falls $x \in L$, und $A_r(x) = 0$, falls $x \notin L$. Wir können nun A_r verwenden, um einen Aufzählungsalgorithmus A_z für L zu entwerfen. Der Algorithmus A_z zählt für die Eingabe $n \in \mathbb{N} - \{0\}$ einfach in einer Schleife die Wörter aus Σ^* in kanonischer Reihenfolge auf und testet für jedes Wort mit Hilfe von A_r, ob es in L liegt. Falls ja, wird das Wort ausgegeben und ein Zähler um 1 erhöht, sonst nicht. Sobald n Wörter ausgegeben wurden, terminiert A_z. Offenbar zählt A_z damit genau die kanonisch ersten n Wörter aus L auf.

Sei nun umgekehrt A_z ein Aufzählungsalgorithmus für eine unendliche Sprache L über einem Alphabet Σ. Dann können wir A_z verwenden, um einen Algorithmus A_r zu entwerfen, der L erkennt. Für ein gegebenes Wort $x \in \Sigma^*$ berechnet A_r die Position n von x in der kanonischen Reihenfolge aller Wörter und ruft dann den Aufzählungsalgorithmus A_z auf der Eingabe n auf. Damit gibt A_z die kanonisch ersten n Wörter aus L aus, betrachtet also auf jeden Fall mindestens die kanonisch ersten n Wörter aus Σ^* und damit auch x. Falls das Wort x in der Ausgabe von A_z enthalten ist, gibt A_r die Antwort 1, weil dann $x \in L$ gilt. Ansonsten gibt A_r die Antwort 0, weil dann $x \notin L$ gilt.

Lösung zu Aufgabe 5

Das Intervall $[2^n, 2^{n+1} - 1]$ enthält 2^n natürliche Zahlen der binären Länge $n + 1$. Weil die binären Darstellungen dieser Zahlen alle mit einer führenden 1 beginnen, kann man sie identifizieren mit den Wörtern der Länge n über dem binären Alphabet Σ_{bool}. Es bleibt also zu zeigen, dass es unter den 2^n Wörtern der Länge n mindestens $2^n - 2^{n-i}$ Wörter w gibt mit $K(w) \geq n - i$.

Die Anzahl der unterschiedlichen Programme der Länge kleiner als $n - i$ ist höchstens

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

da es für jede Länge nicht mehr Maschinencodes als nichtleere Binärstrings geben kann. Also gibt es maximal $2^{n-i} - 2$ Wörter w mit $K(w) < n - i$. Für die Anzahl m der Wörter w in $(\Sigma_{\text{bool}})^n$ mit $K(w) \geq n - i$ gilt also

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

Lösung zu Aufgabe 6

(a) Wir geben zunächst für jedes $n \in \mathbb{N} - \{0\}$ ein Pascal-Programm an, das w_n erzeugt:

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

Dieses Programm berechnet zunächst $k = 2n^4$ und verwendet dann eine Schleife, um $j = 2^k = 2^{2n^4}$ zu berechnen. In einer weiteren Schleife berechnet es dann $l = 2^j = 2^{2^{2n^4}}$. Anschliessend gibt es in der dritten Schleife $l = 2^{2^{2n^4}}$ Nullen aus.

Wenn wir die Operation ^ für die Exponentiation zulassen (dies ist nicht ursprüngliche Pascal-Syntax, wir wollen diese Notation aber im Folgenden verwenden, um die Programme lesbarer darzustellen), erzeugt auch das folgende Programm das Wort w_n:

```
begin
  k:= n;
  k:= 2*k^4;
  k:= 2^(2^k);
  for i:=1 to k do
    write(0);
end.
```

Der einzige Teil des Maschinencodes dieses Programms, der von w_n abhängt, ist die Darstellung von n in der zweiten Zeile. Der restliche Programmcode hat eine konstante Länge. Also ist die binäre Länge dieses Programms $\lceil \log_2(n+1) \rceil + c$ für eine Konstante c.

Damit lässt sich die Kolmogorov-Komplexität von w_n von oben abschätzen durch

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

für eine Konstante c und $c' = c + 1$.

Die Länge von w_n ist $|w_n| = 2^{2^{2n^4}}$. Also gilt $\log_2 |w_n| = 2^{2n^4}$ und damit

$$\begin{aligned} \log_2 n &= \log_2(\sqrt[4]{\tfrac{1}{2} \log_2 \log_2 |w_n|}) = \tfrac{1}{4}(\log_2(\tfrac{1}{2} \log_2 \log_2 |w_n|)) \\ &= \frac{1}{4}(\log_2 \log_2 \log_2 |w_n|) - 1) \,. \end{aligned}$$

Es ergibt sich eine Schranke von

$$K(w_n) \leq \tfrac{1}{4} \log_2 \log_2 \log_2 |w_n| + c''$$

für die Kolmogorov-Komplexität von w_n, wobei $c'' = c' - \frac{1}{4}$ eine Konstante ist.

(b) Wir definieren die Zahlenfolge $y_1, y_2, y_3, \ldots$ durch $y_i = 2^{2^{i+2}}$ für alle $i \in \mathbb{N} - \{0\}$. Offenbar gilt $y_i < y_{i+1}$ und $i = \log_2 \log_2 \sqrt[4]{y_i}$ für alle i. Die Binärdarstellung von y_i ist eine Eins gefolgt von 2^{i+2} Nullen, also $\mathrm{Bin}(y_i) = 1(0)^{2^{i+2}}$.

Nun konstruieren wir für jedes y_i ein Pascal-Programm C_i, das $\mathrm{Bin}(y_i)$ erzeugt:

```
begin
  k:= 2^(i+2);
  write(1);
  for j:=1 to k do
    write(0);
end.
```

In diesem Programm wird zuerst der Wert $k = 2^{i+2}$ berechnet und anschliessend werden in der `for`-Schleife entsprechend viele Nullen in der Ausgabe erzeugt. Der Teil des Maschinencodes von C_i, der von y_i abhängt, ist nur die Darstellung von i. Der Maschinencode für die Darstellung des restlichen Programmteils hat also nur eine konstante Länge, während die binäre Kodierung von i die Länge $\lceil \log_2(i+1) \rceil \leq \lceil \log_2 i \rceil + 1$ hat. Damit folgt

$$\begin{aligned} K(y_i) &\leq \lceil \log_2 i \rceil + c \\ &\leq \lceil \log_2 \log_2 \log_2 \sqrt[4]{y_i} \rceil + c \end{aligned}$$

für eine Konstante c.