Informatik ⊏: – Blatt 9

Rasmus Diederichsen

8. Juli 2014

Aufgabe 9.1

Das Wort “nichtselbstidentifizierend” ist weder selbstidentifizierend noch nichtselbstidentifizierend. Es kann nicht unter den Begriff *selbstidentifizierend* fallen, da es nicht mit dem Namen des Begriffs (“selbstidentifizierend”) übereinstimmt. Es kann nicht nichtselbstidentifizierend sein, denn dann fiele es unter den Begriff *nichtselbstidentifizierend*, wäre also zugleich selbstidentifizierend und nichtselbstidentifizierend. Dies ist ein Widerspruch.

Aufgabe 9.2

Sei das Eingabealphabet von M bezeichnet mit Γ. Ein Algorithmus, der für gegebene M, Z_i die positive Hälfte entscheidet, ob es eine Eingabe w gibt, auf der M Z_i erreicht, kann folgendermaßen aussehen:

```
for n in range(0,∞):
   for w in Γ^{k≤n}:
      lasse M mit Eingabe w n Schritte ausfuehren
      if M erreicht Z_i:
         return 1
```

Aufgabe 9.3

Wir definieren eine Funktion als Programm $\mathcal{P}_1$ mit

$$\mathcal{P}_1(n) = \begin{cases} 1 & \text{falls Eingabe } n \text{ in einen Zyklus 4, 2, 1 führt} \\ undef & \text{sonst} \end{cases}$$

$\mathcal{P}_1$

```
int P_1(int n)
{
   if (n < 1) return 1 // P sollte für alles anhalten, auch wenn
        collatz(n) nicht definiert ist
   if (collatz(n) == 4) // wenn Eingabe 4, dann wiederholt sich
       4, 2, 1
```

```
5           return 1
6       else return P1(collatz(n))
7   }
```

Falls $\mathcal{A}(\mathcal{P}_1) = 1$, so ist die Collatz-Vermutung bewiesen.

Aufgabe 9.4

Angenommen, es gäbe ein Programm $\mathcal{P}$, das für eine gegebene Turingmaschine entscheidet, ob sie jemals anhält.

$\mathcal{P}$

```
1 boolean haeltNicht(TuringMachine t)
2 {
3     ...
4 }
```

Wir definieren das Programm $\mathcal{P}_2$

$\mathcal{P}_2$

```
1 void widerspruch(String input)
2 {
3     if(haeltNicht(new TuringMachine(widerspruch))) return;
4     else while(true);
5 }
```

Liefert die `if`-Abfrage `true`, so hält $\mathcal{P}_2$ für keine Eingabe an, hält dann aber an. Liefert die Abfrage `false`, so hält $\mathcal{P}_2$ für irgendeine Eingabe. Die Eingabe wird aber ignoriert und $\mathcal{P}_2$ gerät in eine Endlosschleife, hält also für keine Eingabe an. Dies ist ein Widerspruch. Das Problem ist also nicht entscheidbar.

Aufgabe 9.5

a)

Eine Lösung ist

we	in	h	ier	un	db	ier	dort
weinh	ier	un	d	b	ier	d	ort

b)

Es ist keine Lösung möglich. Es gilt $\forall i : |y_i| \geq |x_i| \wedge \sum_i |x_i| < \sum_i |y_i|$. Es muss jedoch für eine Lösung gelten $\sum_k |x_{i_k}| = \sum_k |y_{i_k}|$. Dies ist nicht erfüllbar, weil man aufgrund der ersten Beobachtung ein durch y_is generiertes Längedefizit nicht durch Wiederholung eines x_is kompensieren kann. Die beiden Zeilen können also niemals die gleiche Länge haben, es sei denn, man verwendet nur die Steine, die oben wie unten die gleiche Zeichenzahl besitzen. Dies sind

Offensichtlich kann man diese beiden auf keine Weise arrangieren, die die Anforderung erfüllt.

Aufgabe 9.6

a)

Die Rückgabe ist immer korrekt, weil $\mathcal{K}$ alle möglichen PURL-Programme durchprobiert, bis eins die Ausgabe w erzeugt. Da hierbei von kurz nach lang iteriert wird, kommt hier zudem das kürzestmögliche heraus. Es gilt $\mathcal{K}(w) \leq |w| + \lfloor \log_{10} |w| \rfloor + 2$, da eine Ausgabe w in jedem Fall durch das Programm $|w|\texttt{:}w$ generiert werden. $\log_{10} |w| + 1$ liefert gerade die Anzahl von Ziffern der dezimalkodierten Länge von w. Hinzu kommen noch "`:`" sowie alle Zeichen des Wortes selbst. Mithin ist maximal diese Programmlänge nötig.

b)

$\mathcal{K}$ terminiert immer, sofern $|w| < \infty \wedge |\Sigma| < \infty$, da beide Schleifenindizes dann endliche Wertebereiche haben und ein PURL-Programm keine Endlosschleifen enthalten kann.

c)

Die Nichtberechenbarkeit der Kolmogorov-Komplexität gilt nur für Turing-vollständige Sprachen. PURL ist dies nicht. Die Sprache erlaubt keine Endlosschleifen und es gibt keine Variablen oder arithmetische Operationen und keine Verzweigung. Es ist daher nicht möglich, alle berechenbaren Funktionen in PURL zu schreiben.