ETH Zürich
Institut für Theoretische Informatik
Prof. Dr. Angelika Steger, Dr. Johannes Lengler
Übungsleitung: Florian Meier

HS 2019

Algorithmen und Komplexität
Lösungsvorschlag zu Übungsblatt 10

Lösungsvorschlag zu Aufgabe 1

(a) Die Elemente in $\{0,1\}^*$ sind Wörter endlicher Länge, bestehend aus den Zeichen 0 und 1. Es genügt, die Wörter nach ihrer Länge zu ordnen und Wörter gleicher Länge lexikographisch (bezüglich der Binärdarstellung) zu ordnen: 0 sei das erste Wort, 1 das zweite, 00 das dritte, 01 das vierte, usw. Da jedes Wort x endliche Länge hat, wird es irgendwo in dieser Liste auftauchen. Damit gilt insbesondere

$$\{0,1\}^* = \{x_1, x_2, \ldots\}.$$

(b) Programme lassen sich als endliche Sequenz von Zeichen aus einem endlichen Alphabet schreiben. Auch hier können wir die Programme nach ihrer Länge ordnen und Programme gleicher Länge lexikographisch anordnen. Daraus ergibt sich eine Liste

$$\mathcal{P} = \{P_1, P_2, \ldots\}.$$

in der jedes Program enthalten ist, da jedes Programm endliche Länge hat, und somit nur endlich viele Programme vor ihm in der Liste auftauchen.

(c) Wir definieren die Funktion $f_{diag} : \{0,1\}^* \to \{0,1\}$ folgendermassen. Falls $P_i(x_i) \neq 0$ so definiere $f_{diag}(x_i) = 0$, und falls $P_i(x_i) = 0$ so definiere $f_{diag}(x_i) = 1$. Dann gilt für jedes P_i, dass $P_i(x_i) \neq f_{diag}(x_i)$. Daraus folgt, dass f_{diag} von keinem Programm P_i berechnet wird. Deshalb ist f_{diag} nicht berechenbar.

Lösungsvorschlag zu Aufgabe 2

a) Der Algorithmus B bekommt (c,k) als Input. Er setzt $a \leftarrow c$, sortiert c und setzt anschliessend $b \leftarrow c$. Anschliessend ruft er $A(a,b,k)$ auf.

b) Die Korrektheit von B folgt direkt aus der Aussage, dass c genau dann eine aufsteigende Teilsequenz der Länge k besitzt, wenn a und b eine gemeinsame Teilsequenz der Länge k besitzt. Wir zeigen beide Richtungen.

Angenommen c hat eine aufsteigende Teilsequenz x der Länge k. Sicherlich ist x Teilsequenz von a da $a = c$. Da b alle Zahlen von x enthält und sowohl x als auch b aufsteigend sortiert sind, ist x auch Teilseqeuenz von b. Es folgt a und b haben eine gemeinsame Teilsequenze der Länge k.

Angenommen a und b haben eine gemeinsame Teilsequenz x der Länge k. Da b aufsteigend sortiert ist, ist auch x aufsteigend sortiert. Da $c = a$ ist, hat c ein aufsteigende Teilsequenz der Länge k.

c) Algorithmus B benötigt Zeit $n \log n$ um den Input (a,b,k) zu erzeugen (Sortieren mit MERGE-SORT) und zusätzlich Laufzeit $T(n)$ um A aufzurufen. Es folgt Gesamtlaufzeit $\mathcal{O}(n \log n + T(n))$.

Lösungsvorschlag zu Aufgabe 3

a) Wir nehmen per Widerspruch an, dass die Funktion H_0 berechenbar ist. Unser Ziel ist es zu zeigen, dass dann auch die Haltefunktion H berechenbar wäre, was aber gemäss Satz 5.3 ausgeschlossen ist. Sei also H_0 berechenbar. Per Definition existiert dann ein Programm P_0, welches H_0 berechnet. Daraus konstruieren wir ein Programm P, welches die Halte-Funktion H berechnen kann.

Sei (A, x) die Eingabe, die unser Programm P als Bitstring erhält. Wir definieren nun ein Hilfs-Programm B_x wie folgt: B_x ignoriert die Eingabe z und führt das Programm A auf Eingabe x aus. B_x simuliert dann also A auf x. Wir stellen fest, dass $B_x(0) = A(x)$, falls A auf x hält. Falls A auf x *nicht* hält, so hält auch B_x auf Eingabe 0 nicht.

Unser Programm P soll jetzt wie folgt aussehen: Zunächst überprüft P die Eingabe (A, x). Falls A kein Programm beschreibt, gibt P sofort 0 aus. Andernfalls modifiziert P die Codierung von A so, dass B_x entsteht. Dies ist möglich indem wir x als Eingabe hart codieren (In RAM-Maschinen befindet sich die Eingabe in M_0, dann würden wir einfach am Anfang von A den Befehl $M_0 \leftarrow x$ hinzufügen, was soviel bedeutet dass jede Eingabe durch x ersetzt wird). Schliesslich führt P das Programm P_0 auf Eingabe B_x aus, d.h. P berechnet $P_0(B_x)$ und übernimmt $P_0(B_x)$ als *eigene* Ausgabe. Wir behaupten, dass unser Programm P die Haltefunktion berechnet.

- Gilt $P(A, x) = 1$, so stellt einerseits A ein Programm dar, und andererseits $1 = P(A, x) = P_0(B_x)$. Das bedeutet wiederum, dass B_x auf Eingabe 0 hält. Per Konstruktion von B_x macht B_x auf Eingabe 0 genau dasselbe wie A auf Eingabe x, folglich hält dann auch A auf x. Genau das ist, was wir haben wollen: P soll testen, ob A auf x hält. Das heisst, in diesem Fall haben wir $P(A, x) = H(A, x)$.
- Gilt hingegen $P(A, x) = 0$, so ist entweder A nicht die Codierung eines Programms, oder wir haben $0 = P(A, x) = P_0(B_x)$. Dies impliziert, dass B_x auf Eingabe 0 nicht hält. Nach Konstruktion von B_x hält dann Programm A nicht auf Eingabe 0. Also haben wir wieder $P(A, x) = H(A, x)$.

Damit haben wir also ein Programm P geschrieben, welches die Haltefunktion H berechnen kann. Dies steht aber in direktem Widerspruch zu Satz 5.3. Also kann es kein Programm P_0 geben, welches H_0 berechnet.

b) Wir gehen analog vor, wie beim Beweis aus der Vorlesung für den ersten Unvollständigkeits Satz.

Nehme an dass für jedes Programm A entweder $A(0)$ hält und die Aussage "$A(0)$ hält" beweisbar ist, oder $A(0)$ nicht hält und die Aussage "$A(0)$ hält nicht" beweisbar ist. Dann können wir folgendes Programm definieren, welches die Funktion H_0 berechnet:

```
Eingabe: A
if A kein Programm then
  return 0
else
  for all p ∈ {0,1}* do
    if V( "A(0) hält", p)= 1 then
      return 1
    end if
    if V("A(0) hält nicht", p)= 1 then
      return 0
    end if
  end for
end if
```

Falls $H_0(A) = 1$ so hält $A(0)$. Per Annahme findet dieses Programm irgendwann einen Beweis p und es wird 1 ausgegeben. Falls $H_0(A) = 0$ so hält $A(0)$ nicht. Auch in diesem Fall findet per Annahme das Programm einen Beweis p und es wird 0 ausgegeben. Somit berechnet das Program die Funktion H_0. Widerspruch, und die Aussage folgt.