

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T1

Entwerfen Sie eine Turingmaschine, welche genau die Palindrome über dem Alphabet $\{0, 1\}$ akzeptiert.

Aufgabe T2

Zeigen Sie, daß jede 1-Band TM durch eine 1-Band TM mit einseitig unendlichem Band, d.h. durch eine Turingmaschine, die die Positionen $p < 0$ nie benutzt, simuliert werden kann. Wie groß ist der Zeitverlust?

Aufgabe T3

Ein Graph heißt zusammenhängend, wenn jeder Knoten von jedem anderen Knoten durch einen Pfad erreichbar ist. Die Sprache $L_{\text{connected}}$ enthalte alle Kodierungen aller zusammenhängenden Graphen.

Geben Sie eine formale Darstellung für $L_{\text{connected}}$ an. Machen Sie sich dabei insbesondere Gedanken zur Kodierung der Eingabe, zur Eingabelänge und zum Eingabealphabet.

Aufgabe H1 (10 Punkte)

Geben Sie eine Beschreibung des Verhaltens der folgenden Turingmaschiner M . Hält M auf allen Eingaben? Falls ja, welche Sprache wird von M entschieden?

$$M = (\{q_0, q_1, q_2, q_3, q_4, \bar{q}\}, \{0, 1\}, \{0, 1, B\}, B, q_0, \bar{q}, \delta)$$

δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	(q_3, B, R)
q_1	(q_2, B, R)	(q_1, B, R)	(q_3, B, R)
q_2	(q_0, B, R)	(q_4, B, R)	(q_3, B, R)
q_3	(q_3, B, R)	(q_3, B, R)	$(\bar{q}, 0, N)$
q_4	(q_4, B, R)	(q_4, B, R)	$(\bar{q}, 1, N)$

Aufgabe H2 (10 Punkte)

Beschreiben Sie formal eine Turingmaschine, die die Sprache $\{q \in \{0, 1\}^* \mid |w|_0 = |w|_1\}$ entscheidet.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T3

Geben Sie die Gödelnummer $\langle M \rangle$ der unten angegebenen Turingmaschine an.

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, B, q_1, q_3, \delta)$$

δ	0	1	B
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, 0, L)$
q_2	$(q_2, 0, L)$	$(q_2, 1, L)$	(q_3, B, R)

Nutzen Sie dabei die Definition der Gödelnummer aus der Vorlesung.

Aufgabe T4

Wiederholen Sie kurz das Konzept der universellen Turingmaschine.

Es sei U die universelle Turingmaschine. Wie arbeitet U auf der Eingabe $\langle U \rangle \langle U \rangle \langle U \rangle \langle M \rangle w$? Dabei sind M eine beliebige Turingmaschine und w ein beliebiges Eingabewort.

Aufgabe T5

Sind die folgenden Sprachen rekursiv? Begründen Sie Ihre Antwort.

- $L_{100} = \{ \langle M \rangle \mid M \text{ hält auf der leeren Eingabe in höchstens 100 Schritten} \}$.
- $L'_{100} = \{ \langle M \rangle \mid M \text{ besucht höchstens 100 Bandplätze auf der leeren Eingabe} \}$.

Aufgabe H3 (15 Punkte)

Es ist sehr nützlich über einen Turingmaschinensimulator zu verfügen, wenn man Turingmaschinen entwirft. Ziel dieser Aufgabe ist es, einen solchen Simulator zu implementieren. Wir wollen genau solche Turingmaschinen simulieren, wie sie in der Vorlesung definiert wurden. Der Simulator soll als Eingabe einen Dateinamen einer Datei erhalten, welche eine Beschreibung der zu simulierenden Turingmaschine $M = \{Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta\}$ enthält, und ein Eingabewort $w \in \Sigma^*$.

Der Simulator soll dann M auf der Eingabe w simulieren und alle durchlaufenen Konfigurationen in jeweils einer Zeile ausgeben.

Die Beschreibung einer Turingmaschine in einer Datei ist dabei folgendermaßen aufgebaut:

- Die erste Zeile enthält die Anzahl der Zustände n , wobei wir $Q = \{q_1, q_2, \dots, q_n\}$ voraussetzen.
- Die zweite Zeile enthält alle Zeichen aus Σ .

3. Die dritte Zeile enthält alle Zeichen aus Γ .
4. Die vierte Zeile enthält die Nummer i des Anfangszustands $q_i = q_0$.
5. Die fünfte Zeile enthält die Nummer j des Endzustands $q_j = \bar{q}$.
6. Die folgenden Zeilen enthalten je einen Übergang $\delta: (q, a) \mapsto (q', b, D)$ der Übergangsfunktion δ , wobei q, a, q', b und D in dieser Reihenfolge durch jeweils ein Leerzeichen getrennt auftreten.
7. Wir legen fest, daß das Blanksymbol B ist und nicht undefiniert wird.

Folgende Vereinbarungen vereinfachen die Implementierung und Verwendung des Simulators etwas: Sie müssen nicht überprüfen, ob die Beschreibung und das Eingabewort syntaktisch und semantisch korrekt sind. Füttert man den Simulator mit falschen Daten, dann darf etwas beliebiges passieren. Der Simulator muß daher auch nicht alle Zeilen der Eingabe berücksichtigen. Es ist beispielsweise erlaubt, die Zeilen mit Σ und Γ zu ignorieren und nur die Beschreibung von δ zu verwenden. Es ist auch erlaubt, daß δ nicht vollständig angegeben wird und einige Übergänge fehlen. Das macht zum Beispiel dann Sinn, wenn Sie wissen, daß diese Übergänge sowieso nie verwendet werden. So läßt sich viel Schreibarbeit einsparen.

Sie müssen Ihr Programm nicht hocheffizient optimieren, aber es sollte auch nicht zu langsam sein. Längere Simulationen sollten mit nur kurzer Wartezeit durchgeführt werden können. Verwenden Sie eine vernünftige Programmiersprache, die halbwegs bekannt ist und deren Programme von den Tutoren verstanden werden (Java, C, C++, usw. sind alle sehr dafür geeignet. Javascript wäre wohl keine gute Idee).

Zum Schluß läßt sich das gewünschte Verhalten am besten an einem kleinen Beispiel erläutern:

\$ head add.tm	[1]10#1	B11#0[1]B
7	B1[1]0#1	B11#[2]0B
01#	B10[1]#1	B11[2]#1B
01#B	B10#[1]1	B11#[5]1B
1	B10#1[1]	B11#B[5]B
7	B10#[2]1B	B11#[5]BB
1 0 1 0 R	B10#[3]0B	B11[5]#BB
1 1 1 1 R	B10[3]#0B	B1[6]1BBB
1 # 1 # R	B1[4]0#0B	B[6]11BBB
1 B 2 B L	B1[1]1#0B	[6]B11BBB
2 1 3 0 N	B11[1]#0B	BB[7]11BBB
\$./tm add.tm 10#1	B11#[1]0B	\$

Testen Sie Ihren Simulator an verschiedenen kleinen Turingmaschinen.

Erläutern Sie kurz, wie Ihr Simulator funktioniert und geben Sie Protokolle kleiner Beispielläufe und den Quelltext mit ab.

Aufgabe H4 (8 Punkte)

Entwerfen Sie eine Turingmaschine, welche zwei durch # getrennte, binärkodierte Zahlen entgegennimmt und als Ausgabe ihre binärkodierte Summe präsentiert.

Führen Sie mehrere Simulationen dieser Turingmaschine mithilfe des Simulators aus Aufgabe H3 an aussagekräftigen Beispieleingaben durch.

Natürlich addiert eine solche Turingmaschine relativ langsam, wenn es sich um sehr große Zahlen handelt:

```
$ time ./tm add.tm 110110101011010110#101001110110010 | tail -1  
BB[7]111011111010001000BBBBBBBBBBBBBBBB
```

```
real 0m0.451s  
user 0m0.444s  
sys 0m0.028s  
$
```

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T6

1. Zeigen Sie, daß jede RAM durch eine RAM mit einer festen Anzahl von Registern simuliert werden kann.

Hinweis: Als Zwischenschritt können Sie auch eine RAM durch eine Turingmaschine simulieren.

2. Wir wollen zeigen, dass der Befehlssatz der RAM für die Simulation auf die Befehle LOAD, CLOAD, STORE, CADD, CSUB, GOTO, IF $c(0) \neq 0$ GOTO und END eingeschränkt werden kann. Hierzu nutzen wir aus, daß wir nur eine RAM mit konstant vielen Registern simulieren müssen.

Zeigen Sie, wie die Befehle ADD i und INDLOAD i durch den eingeschränkten Befehlssatz ersetzt werden können.

Aufgabe T7

Zeigen sie mithilfe der Unterprogrammtechnik, daß die Sprachen

1. $A = \{\langle M \rangle w \mid M \text{ akzeptiert } w\}$.
2. $A_{EQ} = \{\langle M_1 \rangle \langle M_2 \rangle \mid L(M_1) = L(M_2)\}$.

nicht entscheidbar sind. Verwenden Sie *nicht* den Satz von Rice.

Aufgabe H5 (5 Punkte)

Führen Sie den zweiten Teil der Aufgabe T6 für den Befehl IF $c(0) < x$ GOTO j durch.

Aufgabe H6 (3 Punkte)

Beweisen oder widerlegen Sie die folgende Behauptung: Eine RAM, deren Registergröße beschränkt ist (zum Beispiel auf 32 bit) kann jede Turingmaschine simulieren.

Aufgabe H7 (9 Punkte)

Zeigen Sie mithilfe der Unterprogrammtechnik, daß die Sprache

$$H_1 = \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe und akzeptiert mindestens ein Wort}\}$$

nicht entscheidbar ist. Gehen Sie dabei besonders auf die Korrektheit ein.

Aufgabe H8 (10 Punkte, späterer Abgabetermin)

Geben Sie eine Turingmaschine mit fünf Zuständen und Bandalphabet $\Gamma = \{0, 1, B\}$ an, die auf der leeren Eingabe ϵ terminiert, so daß nach Ende der Berechnung eine möglichst hohe Anzahl an 1en auf dem Band steht.

Testen Sie ihre Konstruktion mit Ihrem Simulator vom zweiten Übungsblatt. Geben Sie die Anzahl der 1en an, die nach der Terminierung auf dem Band stehen.

Für diese Aufgabe haben Sie Zeit bis zum ersten Termin ihres Tutoriums nach den Weihnachtsferien. (Sie dürfen Ihre Lösung natürlich auch früher abgeben.)

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T8

Betrachten Sie die Sprache $L_{Eigen} = \{ \langle M \rangle \mid M \text{ akzeptiert das Wort } \langle M \rangle \}$. Wiederholen Sie zunächst den Beweis aus der Globalübung, der zeigt, daß L_{Eigen} nicht rekursiv ist. Dieser Beweis benutzte nicht den Satz von Rice — beweisen Sie, daß der Satz von Rice überhaupt nicht auf L_{Eigen} anwendbar ist. Zeigen Sie anschließend, daß L_{Eigen} rekursiv aufzählbar ist.

Aufgabe T9

Es sei

$L_{Platz} = \{ \langle M \rangle \mid \text{es existiert } w \in \{0, 1\}^*, \text{ so daß } M \text{ auf } w \text{ mehr als } |w| \text{ Platz benötigt} \}$.

Beweisen Sie, daß L_{Platz} nicht berechenbar ist.

Aufgabe H9 (15 Punkte)

Beweisen oder widerlegen Sie, daß die folgenden Sprachen entscheidbar sind.

1. $L_{q_0} = \{ \langle M \rangle \mid \text{auf keiner Eingabe verläßt } M \text{ ihren Startzustand} \}$
2. $L_{Quadrat} = \{ \langle M \rangle \mid M \text{ berechnet die Funktion } n \mapsto n^2 \}$
3. $L_{q_3} = \{ \langle M \rangle \mid M \text{ besucht auf jeder Eingabe Zustand } q_3 \}$

Hinweis: Die Zustände von M sind aufsteigend durchnummeriert. Jede Turingmaschine mit drei oder mehr Zustände besitzt also auch einen Zustand q_3 .

$$4. L_{1/\sqrt{2}} = \left\{ \langle M \rangle \mid \lim_{n \rightarrow \infty} \frac{|\{w \mid n = |w|, M \text{ akzeptiert } w\}|}{|\{w \mid n = |w|\}|} = \frac{1}{\sqrt{2}} \right\}$$

Zum Zeigen der Entscheidbarkeit skizzieren Sie eine Turingmaschine, die die Sprache entscheidet, oder beschreiben Sie ein formales Entscheidungsverfahren. Um Unentscheidbarkeit zu zeigen, nutzen Sie die Unterprogrammtechnik oder den Satz von Rice aus der Vorlesung.

Alle Turingmaschinen, die eine Gödelnummer besitzen, verwenden das Eingabealphabet $\Sigma = \{0, 1\}$ und das Bandalphabet $\Gamma = \{0, 1, B\}$. Ihre Zustände sind $\{q_1, q_2, \dots, q_n\}$.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T10

Es sei $L(M) = \{ w \mid M \text{ akzeptiert } w \}$. Sind die folgenden Sprachen rekursiv, rekursiv aufzählbar oder keins von beidem. Beweisen Sie ihre Aussage.

1. $L_1 = \{ \langle M \rangle \mid L(M) = \emptyset \}$
2. $L_2 = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$
3. $L_3 = \{ \langle M \rangle \mid |L(M)| \text{ ist endlich} \}$
4. $L_4 = \{ \langle M \rangle \langle M' \rangle \mid L(M) \cap L(M') = \emptyset \}$

Aufgabe T11

Es seien $f, g: \mathbb{Z}^n \rightarrow \mathbb{Z}$ Polynome mit mehreren Variablen.

$$L_{f \leq g} = \{ f, g \mid f \leq g \text{ hat eine ganzzahlige Lösung} \}$$

Ist $L_{f \leq g}$ entscheidbar? Beweisen sie ihre Aussage.

Hinweis: Eine Beispielinstantz wäre etwa $x + (x + 7) \cdot y \leq z^7 - xy + 13$.

Aufgabe H10 (5 Punkte)

Ist das Problem aus Aufgabe T11 rekursiv aufzählbar? Beweisen Sie ihre Aussage.

Aufgabe H11 (5 Punkte)

Es seien $f, g: \{0, 1\}^* \cup \{\perp\} \rightarrow \{0, 1\}^* \cup \{\perp\}$ partielle Funktionen für die $f(\perp) = \perp$ und $g(\perp) = \perp$ gilt. Beweisen oder widerlegen sie:

Es gibt genau dann eine Turingmaschine, die $f \circ g$ berechnet, wenn es Turingmaschinen M und M' gibt, die f und g berechnen.

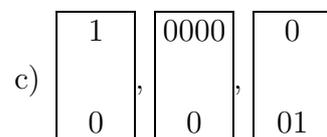
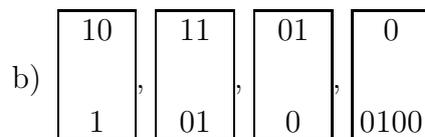
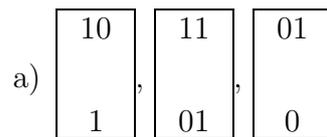
Aufgabe H12 (8 Punkte)

Ist die Sprache $L = \{ \langle M \rangle \langle M' \rangle \mid L(M) \subseteq L(M') \}$ rekursiv aufzählbar? Beweisen Sie ihre Aussage.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T12

Sind die folgenden Instanzen des Postschen Korrespondenzproblems lösbar? Finden Sie eine Lösung oder zeigen Sie die Unlösbarkeit!



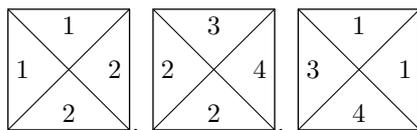
Aufgabe T13

Wenn das Postsche Korrespondenzproblem so modifiziert wird, daß nur ein unäres Alphabet verwendet wird, ist es dann immer noch unentscheidbar?

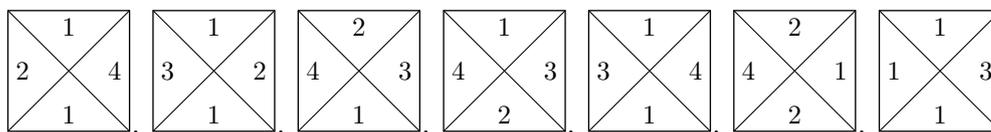
Aufgabe T14

In dieser Aufgabe betrachten wir das folgende Puzzleproblem: Gegeben ist eine Menge von quadratischen Steinen, deren vier Kanten jeweils mit natürlichen Zahlen dekoriert sind. Gesucht ist eine Strategie, eine unendliche Ebene vollständig mit Kopien dieser Steine zu bedecken, wobei aneinanderliegende Kanten stets dieselben Zahlen tragen sollen (es handelt sich gewissermaßen um die unendliche Version des Spiels Tetravex, welches aber, historisch gesehen, selbst als endliche Version unseres Problems entstand).

1. Ist diese Instanz lösbar?



2. Ist diese Instanz lösbar?



Aufgabe H13 (16 Punkte)

Schreiben Sie ein Programm, das eine Instanz des Postschen Korrespondenzproblems über dem Alphabet $\{0,1\}^*$ einlesen kann und anschließend zu lösen versucht. Wir empfehlen, Backtracking mit einem Abbruch bei einer vorgegebenen Suchtiefe zu verwenden. Das Programm soll entweder eine gefundene Lösung ausgeben, sagen, daß es keine Lösung gibt, oder zugeben, daß es unfähig war diese Instanz zu lösen.

Geben Sie einen Ausdruck Ihres Programms ab und Protokolle über die Arbeit Ihres Programms auf den Instanzen aus Aufgabe T12 und den folgenden:

a)

0
1

,

01
0

,

1
101

b)

01
000

,

10
111

,

0
1

,

0
010

,

001
01

,

10
1

,

010
11

Bedenken Sie, daß die Tutoren in der Lage sein müssen, Ihr Programm nachzuvollziehen.

Lassen Sie Ihr Programm *nicht* auf diesen Instanzen laufen:

c)

1101
1

,

0110
11

,

1
110

d)

10
0

,

0
001

,

100
1

e)

101
0

,

1
101

,

0
1

Aufgabe H14 (6 Punkte)

Ist folgende Variante des Postschen Korrespondenzproblems entscheidbar? Wir verlangen, daß das Alphabet $\{0,1\}^*$ ist und jedes Wort auf jeder Karte höchstens die Länge 17 besitzt.

Aufgabe H15 (Bonusaufgabe, sehr viele Punkte)

Ist das Puzzleproblem aus Aufgabe T14 entscheidbar?

Hinweis: Diese Aufgabe ist sehr schwierig.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T15

Entwickeln Sie ein WHILE-Programm, das den Wert 2^{x_1} berechnet. Analysieren Sie die Laufzeit ihres Programmes im uniformen und im logarithmischen Kostenmaß.

Aufgabe T16

Die Ackermannfunktion $A : \mathbb{N}^2 \rightarrow \mathbb{N}$ wurde in der Vorlesung folgendermaßen definiert:

$$\begin{aligned} A(0, m) &= m + 1 && \text{für } m \geq 0 \\ A(n + 1, 0) &= A(n, 1) && \text{für } n \geq 0 \\ A(n + 1, m + 1) &= A(n, A(n + 1, m)) && \text{für } n, m \geq 0 \end{aligned}$$

- Zeigen Sie, daß die Ackermannfunktion für alle Parameter $n, m \in \mathbb{N}$ terminiert.
- Beweisen Sie durch Induktion nach n folgende Aussage:

$$A(n, m) \leq A(n + 1, m - 1) \text{ für alle } m \geq 0$$

Hinweis: Nutzen Sie die Monotonie der Ackermannfunktion in beiden Parametern aus.

Aufgabe T17

Sind WHILE-Programme immer noch Turing-mächtig, wenn die Zuweisungen $x_i := x_j + c$ nur noch für $c \in \{-1, 0\}$ erlaubt sind?

Aufgabe H16 (8 Punkte)

Betrachten Sie die nachfolgenden Varianten des Euklidischen Algorithmus (entnommen aus Wikipedia) zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen a und b .

Variante 1:

```
Eingabe:  $a, b \in \mathbb{N}$   
While  $a \neq b$   
  If  $a > b$   
    then  $a := a - b$   
    else  $b := b - a$   
End While  
Ausgabe:  $a$ 
```

Variante 2:

```
Eingabe:  $a, b \in \mathbb{N}$   
While  $b > 0$   
   $r := a \bmod b$   
   $a := b$   
   $b := r$   
End While  
Ausgabe:  $a$ 
```

1. Bestimmen Sie eine möglichst scharfe untere Schranke für die Worst-Case-Laufzeit von Variante 1 im uniformen Kostenmaß.
2. Bestimmen Sie eine möglichst scharfe obere Schranke für die Worst-Case-Laufzeit von Variante 2 im uniformen Kostenmaß.
3. Nutzen Sie Ihre Abschätzungen, um zu zeigen, daß sich die uniformen Worst-Case-Laufzeiten beider Varianten durch einen exponentiellen Faktor unterscheiden.
4. Stimmt diese Aussage auch bezüglich der Laufzeiten im logarithmischen Kostenmaß?

Aufgabe H17 (6 Punkte)

Geben Sie ein LOOP-Programm an, daß folgende Sprache akzeptiert:

$$L = \{w \in \{0, 1\}^* \mid |w|_0 = |w|_1\}$$

wobei $|w|_i$ die Anzahl der Stellen in w angibt, an denen die Ziffer i steht.

Gehen Sie davon aus, daß die Eingabe in der Variable x_1 als Binärzahl kodiert steht und das zudem die Länge der Eingabe in der Variable x_2 zu finden ist. Wenn der Wert x_1 in der Sprache L enthalten ist, soll am Ende des Programs x_0 eine Eins enthalten, ansonsten Null.

Nehmen Sie an, daß die Subtraktion von Variablen mit dem Wert 0 wiederum 0 ergibt (Variablen können also nie negative Werte enthalten). Zudem setzen wir voraus, daß die Eingabe x_1 nie das leere Wort enthält.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T18

- a) Wiederholen Sie die Definition der O -Notation.
b) Welche der folgenden Aussagen sind wahr?

1. $n^2 = O(2^n)$
2. $\sqrt{n} = O(n/\log n)$
3. $\sqrt{n-1} = \sqrt{n} + O(1)$
4. $\int_0^{O(n)} O(t) dt = O(n^2)$
5. $(\log n)^n = O(n^{\log n})$

Aufgabe T19

Wir bezeichnen mit p_N die N -te Primzahl.

Zu gegebener Eingabe $N \in \mathbb{N}$ sei die kleinste Primzahl gesucht, welche größer als N ist. Ist diese Aufgabe in polynomieller Zeit lösbar, falls

- a) N in unärer Kodierung vorliegt,
b) N in binärer Kodierung vorliegt?

Hinweis zu Teilaufgabe b: Cramérs Vermutung von 1936 besagt, daß die Primzahllücke $g(p_n) = p_{n+1} - p_n$ zwischen der n -ten und $(n+1)$ -ten Primzahl durch

$$g(p_n) = O((\ln p_n)^2)$$

beschränkt ist.

Aufgabe H18 (5 Punkte)

Für einen beliebigen Algorithmus sei $t(n) \geq n$ die Laufzeit im uniformen Kostenmaß und $t'(n)$ die Laufzeit im logarithmischen Kostenmaß.

Finden Sie eine möglichst interessante Beziehung zwischen $t(n)$ und $t'(n)$.

Aufgabe H19 (10 Punkte)

Welche Funktion berechnet der folgende Algorithmus?

Analysieren Sie seine Laufzeit sowohl im uniformen als auch im logarithmischen Kostenmaß.

Eingabe: $a \in \mathbb{N}$

```
b = 2
while a ≠ 0
  b = b · b
  a = a - 1
end while
c = 2
while b ≠ 0
  c = c · c
  b = b - 1
end while
```

Ausgabe: c

Aufgabe H20 (8 Punkte)

Als Eingabe sei eine Menge von Intervallen

$$\left\{ \left[\frac{a_1}{b_1}, \frac{c_1}{d_1} \right], \left[\frac{a_2}{b_2}, \frac{c_2}{d_2} \right], \left[\frac{a_3}{b_3}, \frac{c_3}{d_3} \right], \dots, \left[\frac{a_n}{b_n}, \frac{c_n}{d_n} \right] \right\}$$

gegeben.

Die Anfangs- und Endpunkte jedes Intervalls sind rationale Zahlen, welche durch die Binärdarstellung ihres Zählers und Nenners kodiert werden.

Ist in polynomieller Zeit berechenbar, ob der Schnitt zwischen allen gegebenen Intervallen leer ist?

Falls Sie der Meinung sind, daß die Antwort *ja* ist, dann geben Sie einen Algorithmus an, der die Aufgabe in polynomieller Zeit löst. Wie groß ist die Laufzeit Ihres Algorithmus?

Aufgabe H21 (Bonusaufgabe)

Gegeben sind zwei Mengen von natürlichen Zahlen $\{k_1, \dots, k_n\}$ und $\{l_1, \dots, l_n\}$, die jeweils als durch Trennzeichen separierte Binärdarstellungen als Eingabestring repräsentiert werden.

Kann die folgende Frage in polynomieller Zeit beantwortet werden?

$$\sqrt{k_1} + \dots + \sqrt{k_n} < \sqrt{l_1} + \dots + \sqrt{l_n}$$

Hinweis: Vorsicht! Dies ist eine seit vielen Jahren offene Frage, deren Beantwortung weitreichende Konsequenzen hätte. Es ist heute noch nicht einmal bekannt, ob dieses Problem in *NP* liegt.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T20

Beweisen Sie die folgende Aussage: Die Entscheidungsvariante von BIN PACKING ist genau dann in polynomieller Zeit lösbar, wenn dies auch für die Optimierungsvariante gilt.

Aufgabe T21

Wir betrachten folgendes Problem: Als Eingabe sind ganzzahlige kartesische Koordinaten von n Punkten in der Ebene sowie weitere Zahlen k und A gegeben. Die Frage ist, ob es k Kreise in der Ebene gibt, welche zusammen alle n Punkte umfassen und deren Flächensumme höchstens A ist.

Beschreiben Sie, wie eine nichtdeterministische Turingmaschine dieses Problem in polynomieller Zeit lösen kann?

Aufgabe T22

Wir betrachten das Problem SORTING BY REVERSALS. Eine gegebene Permutation π soll durch höchstens k viele Vertauschungen (*reversals*) ρ in eine zweite gegebene Permutation σ verwandelt werden. Ein *reversal* $\rho(i, j)$ vertauscht dabei die Reihenfolge aller Elemente im Intervall $[i, j]$. Formal

$$\rho(i, j): (\pi_1 \dots \pi_{i-1} \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n) \mapsto (\pi_1 \dots \pi_{i-1} \pi_j \pi_{j-1} \dots \pi_{i+1} \pi_i \pi_{j+1} \dots \pi_n)$$

Geben Sie ein Zertifikat und einen Polynomialzeitverifizierer an. Beschreiben Sie dazu im Detail die Kodierung und die Länge des Zertifikates und die Arbeitsweise des Verifizierers. Die Motivation dieses Problems kommt aus der Bioinformatik: Viele Mutationen auf DNA-Strängen finden ausschließlich durch solche *reversals* statt. Eine kürzeste Folge dieser reversals entspricht wahrscheinlich den Mutationen, die wirklich stattfanden.

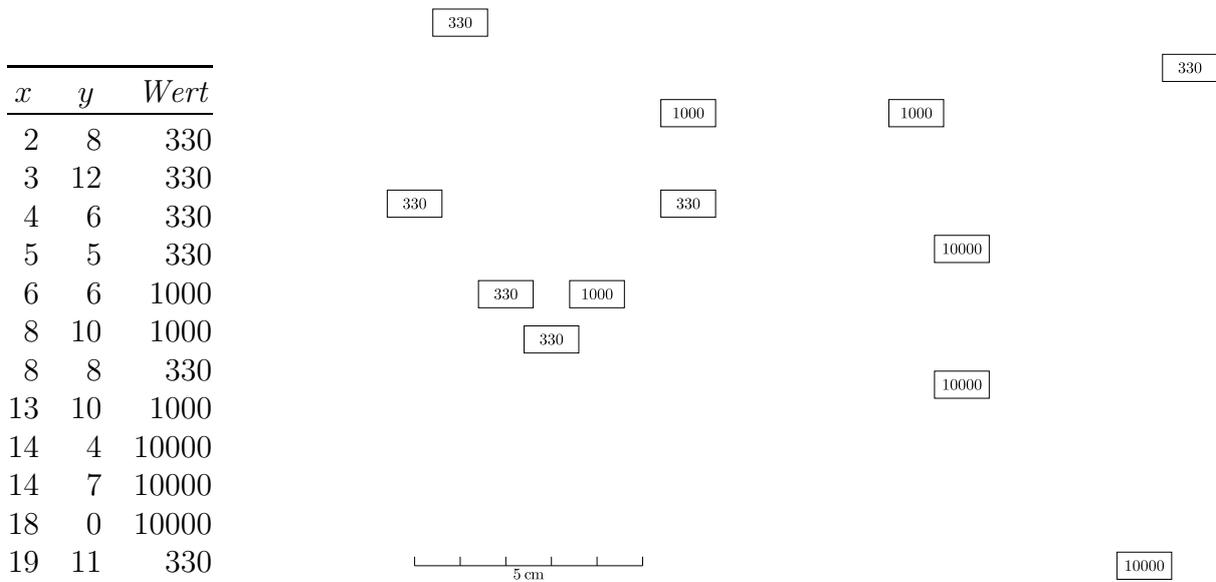
Aufgabe H22 (10 Punkte)

Beweisen Sie die folgende Aussage: Die Entscheidungsvariante des TRAVELLING SALESMAN PROBLEMS (TSP) ist genau dann in polynomieller Zeit lösbar, wenn dies auch für die Optimierungsvariante gilt. Die Gewichte mögen durch rationale Zahlen in Binärkodierung kodiert sein.

Aufgabe H23 (15 Punkte)

Eine selbstgebaute Bestückungsmaschine kann Widerstände mit verschiedenen Widerstandswerten an die richtigen Plätze in eine Platine stecken. Sie benötigt 2 Sekunden, um auf einen verschiedenen Wert zu wechseln, ansonsten kann sich der Bestückungskopf mit einem Zentimeter pro Sekunde in eine beliebige Richtung bewegen. Das eigentliche Einstecken eines Widerstands benötigt ebenfalls etwas Zeit, aber diese Zeit ist fest und hängt insbesondere nicht vom Widerstandswert ab oder davon, was die Maschine in der Vergangenheit machte.

Die folgende Tabelle zeigt die Positionen (in Zentimetern) und Widerstandswerte (in Ohm) von elf Widerständen, mit welchen eine Platine bestückt werden muß. Sie können davon ausgehen, daß der Bestückungskopf sich anfangs bereits auf einer der zu bestückenden Positionen befindet und mit dem richtigen Widerstandswert geladen ist. Der erste Widerstand kann also sofort eingesteckt werden. Die Zeichnung neben der Tabelle zeigt, wie die Widerstände am Ende auf der Platine positioniert sein werden (aber nicht im Maßstab 1 : 1).



- Finden Sie eine optimale Reihenfolge, in welcher alle Widerstände bestückt werden können und der Bestückungskopf anschließend wieder in die Ausgangslage zurückkehrt und wieder mit dem gleichem Widerstandswert wie am Anfang geladen wird. (So kann eine Serie von Platinen nacheinander auf die gleiche Weise bestückt werden.)
- Wie können Sie garantieren, daß Ihre Lösung optimal ist?
- Schätzen Sie, mit bis zu wievielen Widerständen man Ihre Lösungsmethode noch verwenden könnte, bevor die Laufzeit unpraktikabel groß würde.
- Haben Sie eine Idee, was Sie machen würden, wenn es um hunderte von Widerständen geht?

Hinweis: Natürlich dürfen Sie ein Programm schreiben, welches bei der Lösung dieser Aufgabe hilft.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T21

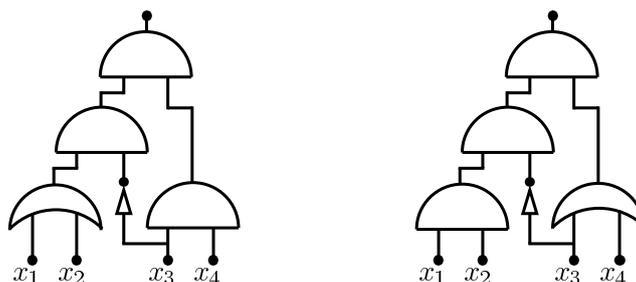
Sie wollen zur Festzeit ihr Haus mit bunten Lampen dekorieren. Leider haben Sie ihren guten Freund, einen Elektrotechnikstudenten, um Hilfe gebeten und nun ist die Beleuchtung derart kompliziert, daß Sie Schwierigkeiten haben, sie überhaupt anzuschalten.

Ein Boolesches Schaltnetz besteht aus einem gerichteten, azyklischen Graphen mit Eingangsknoten x_1, \dots, x_n sowie einem Ausgangsknoten x_{out} . Die Eingangsknoten haben dabei nur eine ausgehende Kanten und der Ausgangsknoten nur eine eingehende Kante. Alle weiteren Knoten sind markiert als ODER-, UND- oder NICHT-Gatter, außerdem sind Verzweigungen möglich.



Für das Entscheidungsproblem **BOOLEAN CIRCUIT** ist ein solches Schaltnetz gegeben und die zugehörige Frage ist, ob es eine erfüllende Belegung gibt.

Die folgenden Schaltungen hat ihr Freund in die Beleuchtung verbaut, mit der Behauptung, daß die jeweilige Lichterkette angeht, wenn man die Schalter x_1, \dots, x_4 in die richtige Stellung bringt. Ist dies tatsächlich für beide Schaltungen möglich? Wenn ja, geben Sie eine Schalterstellung an, die ihr Haus festlich beleuchtet.



Aufgabe T22

Reduzieren Sie das oben beschriebene Problem **BOOLEAN CIRCUIT** das **SAT**-Problem aus der Vorlesung. Formal: Zeigen Sie **BOOLEAN CIRCUIT** \leq_p **SAT**. Denken Sie dabei an Zimt, Tannengeruch und Kerzenschein.

Aufgabe T23

Das Problem **PLANAR BOOLEAN CIRCUIT** ist definiert wie das **BOOLEAN CIRCUIT**, jedoch dürfen sich nun keine zwei Drähte kreuzen. Zeigen Sie:

$$\text{BOOLEAN CIRCUIT} \leq_p \text{PLANAR BOOLEAN CIRCUIT}$$

Denken Sie dabei darüber nach, ob Sie nicht doch noch Geschenke besorgen müssen: wenn Sie dieses Blatt lesen, ist die Zeit schon recht knapp!

Aufgabe H24 (10 Punkte)

Sie müssen die Beleuchtung ihres Hauses reparieren, aber aufgrund einer Knappheit an seltenen Erden sind NICHT-Gatter dieses Jahr ausgesprochen teuer. Sie planen daher, NICHT-Gatter durch schlaue Modifizierung der Schaltung zu ersetzen. Als Hilfsmittel haben sie zudem Schalter, die bereits ein NICHT-Gatter eingebaut haben.

Das MONOTONE BOOLEAN CIRCUIT ist wie BOOLEAN CIRCUIT definiert, aber es darf keine Negationsgatter außer direkt an den Eingängen geben.

Beweisen Sie, daß es eine Möglichkeit gibt, ihre Weihnachtsbeleuchtung ohne NICHT-Gatter zu realisieren, indem Sie zeigen, daß

$$\text{BOOLEAN CIRCUIT} \leq_p \text{MONOTONE BOOLEAN CIRCUIT}$$

Aufgabe H25 (10 Punkte)

Ihr E-Technikerfreund behauptet, er könne Schaltungen mit gleicher Funktion auf einen Blick erkennen. Das Problem BOOLEAN CIRCUIT EQUIVALENCE ist formal wie folgt definiert: als Eingabe erhalten wir zwei Boolesche Schaltkreise und müssen entscheiden, ob die beiden Schaltkreise für alle Eingaben die gleiche Ausgabe liefern.

Zeigen Sie, daß ihr Freund entweder ein Wunderkind oder ein Angeber ist, indem Sie beweisen, daß die folgenden zwei Aussagen gelten:

$$\overline{\text{BOOLEAN CIRCUIT EQUIVALENCE}} \leq_p \text{BOOLEAN CIRCUIT}$$
$$\text{BOOLEAN CIRCUIT} \leq_p \overline{\text{BOOLEAN CIRCUIT EQUIVALENCE}}$$

Aufgabe H26 (10 Punkte)

Der Weihnachtsmann hat m Geschenke, die er an n liebe Kinder verteilen möchte. Jedes Kind kann dabei mehrere Geschenke bekommen, jedoch mag jedes Kind die unterschiedlichen Geschenke unterschiedlich gerne. Ein Kind i möge also ein Geschenk j genau $p_{i,j} \in \mathbb{N}$ gerne (diese hypothetischen Kinder haben äußerst präzise Wunschzettel).

Die Freude eines Kindes i sei dabei als $\mathcal{F}(i) = \sum_{j \in G_i} p_{i,j}$ definiert, wobei $G_i \subseteq \{1, \dots, m\}$ die Geschenke bezeichnet, die das Kind i erhält. Natürlich möchte der Weihnachtsmann allen Kinder eine möglichst glückliche Weihnachtszeit beschern, daher versucht er die Geschenke so in Mengen G_1, \dots, G_n einzuteilen, daß die Freude des traurigsten Kindes maximal ist. Formal er will also die Funktion $c(\mathcal{G}) = \min_{1 \leq i \leq n} \mathcal{F}(i)$ maximieren, wobei \mathcal{G} eine beliebige Partition der m Geschenke bezeichne.

Bei der Entscheidungsvariante des SANTA CLAUS PROBLEM soll entschieden werden, ob eine gegebene Mindestweihnachtsfreude k erreicht werden kann, also, ob es eine Lösung \mathcal{G} gibt, für die die Zielfunktion $c(\mathcal{G}) \geq k$ übersteigt.

Zeigen Sie, daß das SANTA CLAUS PROBLEM NP-vollständig ist.

Hinweis: Für die Reduktion empfehlen wir das NP-vollständige Problem PARTITION, das wie folgt definiert ist:

Gegeben ist eine endliche Menge natürlicher Zahlen $A = \{a_1, \dots, a_n\} \subset \mathbb{N}$. Kann A so in disjunkte Teilmengen B, C zerlegt werden, daß die Summe der Elemente in beiden Teilmengen gleich groß ist?



Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T24

Wir betrachten das folgende Erfüllbarkeitsproblem.

NOT-ALL-EQUAL SAT

Eingabe: Eine aussagenlogische Formel ϕ in 3KNF.

Frage: Gibt es eine erfüllende Belegung, so daß in jeder Klausel mindestens ein wahres und ein falsches Literal vorkommt?

1. Beweisen Sie, daß $3\text{-SAT} \leq_p \text{NOT-ALL-EQUAL-SAT}$
2. Zeigen Sie nun, daß NOT-ALL-EQUAL-SAT auch dann noch NP-vollständig ist, wenn in den einzelnen Klauseln nur positive Literale (also keine negierten Variablen) verwendet werden dürfen.

Aufgabe T25

Das Problem INDUCED CYCLES fragt, ob in einem gegebenen Graphen G mindestens k Knotendisjunkte Kreise existieren, die zudem noch unabhängig sind—d.h. diese Kreise sollen nicht durch Kanten untereinander verbunden sein. Beweisen Sie, daß dieses Problem NP-schwer ist, indem sie eine Reduktion von INDUCED MATCHING angeben (dessen schwere sollen Sie in H27 beweisen).

Aufgabe H27 (15 Punkte)

Eine Kantenmenge $M \subseteq E$ ist ein *Matching* in einem Graphen $G = (V, E)$, wenn für alle Kanten $e_1, e_2 \in M$ gilt, daß $e_1 \cap e_2 = \emptyset$. Ein Matching ist zudem ein *induced Matching*, wenn die Matchingkanten untereinander nicht durch Kanten in G verbunden sind.

Das Problem INDUCED MATCHING besteht nun darin, zu entscheiden ob ein induced Matching der Größe k in einem gegebenen Graphen G existiert. Dieses Problem wird in der Literatur auch als *risk-free marriage problem* bezeichnet: gesucht ist eine Zuteilung auf Ehen, so daß keiner der Ehepartner Interesse an einem Ehepartner einer anderen Ehe hat. Zeigen Sie, daß $\text{INDEPENDENT SET} \leq_p \text{INDUCED MATCHING}$.

Aufgabe H28 (5 Punkte)

Ein *Hamiltonpfad* eines Graphen ist ein Pfad, der alle Knoten des Graphens genau einmal besucht. Das $\text{HAMILTIONPFAD-PROBLEM}$ (HPP) besteht darin, einen solchen Pfad für einen gegebenen Graphen zu finden.

Analog ist beim $1/2\text{-HAMILTIONPFAD-PROBLEM}$ ($1/2\text{-HPP}$) ein Pfad gesucht, der genau die Hälfte der Knoten besucht. Zeigen Sie folgende Beziehung zwischen den beiden Problemen:

$$\text{HPP} \leq_p 1/2\text{-HPP}$$

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T26

Gegen Ende Ihres Studiums haben sich bei Ihnen eine Reihe von Rechnungen aufgetürmt, die Sie nun mit Ihrem knappen Monatsbudget $b \in \mathbb{N}$ begleichen müssen. Jede der n Rechnungen i , die Sie im aktuellen Monat begleichen, kostet Sie einen Betrag von $w_i \in \mathbb{N}$ Euro. Sollten Sie die Rechnung später begleichen, so zahlen Sie einen Betrag von $2w_i$ (der Zeitpunkt ist dann allerdings egal).

Natürlich möchten Sie insgesamt so wenig Geld wie möglich für die Strafgebühren aufwenden: Sie fragen sich also, ob Sie alle Rechnungen so begleichen können, daß insgesamt höchstens $k \in \mathbb{N}$ Gebühren anfallen.

Zeigen Sie, daß das SCHULDEN-Problem NP-schwer ist.

Aufgabe T27

Überraschenderweise finden Sie sich nach erfolgreichem Abschluß Ihres Studiums als Direktor eines Museums für moderne Kunst. Ihre erste Aufgabe ist es, den Rundgang der Nachtwache zu planen. Da die Nachtwache ehrenamtlich durch einen betagten Rentner besetzt ist, ist zwingend notwendig, daß die Strecke des Rundgangs so kurz wie möglich ist. Trotzdem muss jeder letzte Winkel des Museums von diesem Rundgang aus mindestens einmal sichtbar sein. Formal ist das Problem wie folgt definiert:

MUSEUMSNACHTWACHE

Eingabe: Ein Museumsgrundriß M als Menge von Polygonen, eine Zahl $l \in \mathbb{Q}$.

Problem: Gibt es eine Rundtour der Länge höchstens l , von der aus jeder Punkt des Museums sichtbar ist?

Hinweis: Für diese Aufgabe können Sie annehmen, daß METRISCHES TSP NP-vollständig ist.

METRISCHES TSP

Eingabe: Eine Menge von Punkten V in der euklidischen Ebene und eine Zahl $l \in \mathbb{Q}$.

Problem: Gibt es eine Tour der Länge höchstens l , die alle Punkte besucht?

Aufgabe H29 (3+7 Punkte)

Sie haben als Willkommensgeschenk von der Museumsbelegschaft einen Essensgutschein mit Wert k für ein nobles Restaurant bekommen. Das Restaurant hat eine Speisekarte mit n Gerichten. Sie möchten sich in dem Restaurant von der Speisekarte ein Menü zusammenstellen lassen, sodaß der Gutschein möglichst vollständig genutzt wird, ohne daß sie zusätzlich noch etwas zahlen müssen (die Schulden aus Aufgabe T26 zwingen Sie zu Sparsamkeit).

- a) Ist dieses Problem leicht lösbar und wenn ja, wie? Beweisen Sie andernfalls, daß es schwer ist.
- b) Geben Sie einen Algorithmus für das Problem an, der polynomiell ist in n und k . Wieso beweist dieser Algorithmus nicht, daß $P = NP$ ist?

Aufgabe H30 (10 Punkte)

Ihre Anstellung im Museum ist nicht von langer Dauer, ein peinlicher Zwischenfall mit Duchamps *Fountain* bereitete Ihrer Karriere ein unrühmliches Ende. Glücklicherweise finden Sie schnell eine neue Beschäftigung bei "Merkstein Mysteriöse Maschinen".

Sie stehen vor folgender Aufgabe: Ihnen wird die Spezifikation einer Mysteriöse Maschine mit n Zuständen vorgelegt. Eine Mysteriöse Maschine hat eine Reihe von Knöpfen, Rädern und Hebeln, mit denen der interne Zustand verändert werden kann. "Merkstein Mysteriöse Maschinen" liefert qualitativ hochwertige Mysteriöse Maschinen seit 1890, den jede Maschine wird vollständig auf ihre Funktionalität geprüft: dazu muss die Maschine probeweise in jeden der n Zustände versetzt werden. Um den Arbeitsprozess zu optimieren, sollen Sie nun eine möglichst kurze Reihenfolge von Bedienoperationen finden (in der Entscheidungsvariante: höchstens l Operationen), welche die Maschinen in jeden Zustand mindestens einmal versetzt.

Sie vermuten, daß dieses Problem NP-vollständig ist. Finden Sie einen Beweis, um Ihre Vorgesetzten von der Schwere Ihrer Aufgabe zu überzeugen (Ihre Anstellung soll schließlich länger als die vorherige dauern).

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T1

Entwerfen Sie eine Turingmaschine, welche genau die Palindrome über dem Alphabet $\{0, 1\}$ akzeptiert.

Lösungsvorschlag

Es wird jeweils das am linken Rand der Eingabe mit dem am rechten Rand der Eingabe stehenden Symbol verglichen.

Wenn nur ein Band zur Verfügung steht, muss die Turingmaschine für diesen Vergleich einmal über die komplette Eingabe laufen.

1. Falls Eingabe leer, terminiere mit JA.
2. Speichere das aktuell gelesene Symbol im Zustand, und ersetze es durch ein B.
3. Laufe zum rechten Ende der Eingabe, und vergleiche das dort gespeicherte mit dem im Zustand gespeicherten Symbol. Falls ungleich, terminiere mit NEIN. Ansonsten lösche es auf dem Band.
4. Laufe ganz nach links, und gehe zu Schritt 1.

Daraus folgt eine Laufzeit von $\theta(n^2)$ für obigen Algorithmus. Gleiches gilt für die Optimierung, in der beim Zurücklaufen in Schritt 4 gleichzeitig auch ein Symbol vom rechten Rand im Zustand mitgenommen wird, um so bei jedem Lauf über die Eingabe zwei Zeichen miteinander vergleichen zu können.

Aufgabe T2

Zeigen Sie, daß jede 1-Band TM durch eine 1-Band TM mit einseitig unendlichem Band, d.h. durch eine Turingmaschine, die die Positionen $p < 0$ nie benutzt, simuliert werden kann. Wie groß ist der Zeitverlust?

Lösungsvorschlag

Wir simulieren eine 1-Band-TM mit beidseitig unendlichem Band auf einer 1-Band-TM mit einseitig unendlichem Band. Dazu benutzen wir eine zweite Spur. Zugriffe auf Bandpositionen $p \leq 0$ werden auf Position $1 - p$ umgelenkt. An Position 0 steht ein Sonderzeichen, das sonst nicht verwendet wird, z.B. ein \$. Die Beschreibung der Zustandsübergangsfunktion wird dabei doppelt so groß, sie enthält nämlich einmal das ursprüngliche und unveränderte δ , und einmal ein gespiegeltes δ' .

Falls der Kopf auf einer Position $p > 0$ steht, wird Spur 2 ignoriert. Falls Position 0 erreicht ist, was am \$ zu erkennen ist, findet ein Übergang statt, und es wird zum gespiegelten δ'

gewechselt. δ' benutzt Spur 2, und ignoriert Spur 1. Wo δ einen Schritt nach links macht, geht δ' nach rechts, und umgekehrt. Ansonsten bleibt die Berechnung wie bei der TM mit beidseitig unendlichem Band.

Diese Simulation verändert die Laufzeit nicht.

Aufgabe T3

Ein Graph heißt zusammenhängend, wenn jeder Knoten von jedem anderen Knoten durch einen Pfad erreichbar ist. Die Sprache $L_{\text{connected}}$ enthalte alle Kodierungen aller zusammenhängenden Graphen.

Geben Sie eine formale Darstellung für $L_{\text{connected}}$ an. Machen Sie sich dabei insbesondere Gedanken zur Kodierung der Eingabe, zur Eingabelänge und zum Eingabealphabet.

Lösungsvorschlag

Wir kodieren G durch eine Adjazenzmatrix $A = (a_{i,j}) \in \{0, 1\}^{n \times n}$. Für alle Knotenpaare gibt es ein $k \in \{2, \dots, n\}$, so dass ein Weg der Länge k zwischen diesen beiden Knoten existiert.

Sei $A = (a_{ij}) \in \{0, 1\}^{n \times n}$ die Adjazenzmatrix zu G , d.h. $a_{ij} = 1$ genau dann, wenn $\{i, j\} \in E$ ist, für $0 \leq i, j < n$. Die Kodierung von G besteht aus der Aneinanderreihung der Zeilen von A , also aus dem Wort

$$a_{0,0}a_{0,1} \dots a_{0,n-1}a_{1,0} \dots a_{1,n-1} \dots a_{n-1,n-1} \in \{0, 1\}^{n^2}.$$

$$L_{\text{connected}} = \{a_{1,1} \dots a_{n,n} \in \{0, 1\}^{n^2} \mid n \in \mathbb{N}, (\forall i, j \text{ mit } i \neq j) (\exists k \geq 2) (\exists \sigma \in S_n) : \\ \sigma(1) = i \text{ und } \sigma(k) = j \text{ und } \forall 1 \leq l \leq k - 1 : a_{\sigma(l), \sigma(l+1)} = 1\}$$

Aufgabe H1 (10 Punkte)

Geben Sie eine Beschreibung des Verhaltens der folgenden Turingmaschiner M . Hält M auf allen Eingaben? Falls ja, welche Sprache wird von M entschieden?

$$M = (\{q_0, q_1, q_2, q_3, q_4, \bar{q}\}, \{0, 1\}, \{0, 1, B\}, B, q_0, \bar{q}, \delta)$$

δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	(q_3, B, R)
q_1	(q_2, B, R)	(q_1, B, R)	(q_3, B, R)
q_2	(q_0, B, R)	(q_4, B, R)	(q_3, B, R)
q_3	(q_3, B, R)	(q_3, B, R)	$(\bar{q}, 0, N)$
q_4	(q_4, B, R)	(q_4, B, R)	$(\bar{q}, 1, N)$

Lösungsvorschlag

Die Turingmaschine M durchläuft im Anfangszustand q_0 beginnend die Eingabe von links nach rechts. Falls in einem der Zustände q_0 bis q_2 ein Blank gelesen wird, geht M in den Zustand q_3 .

Sobald eine Eins gelesen wird, wechselt M in den Zustand q_1 . Solange weitere Einsen gelesen werden, bleibt M im Zustand q_1 . Ist das nächste Zeichen eine Null, geht M in den Zustand q_2 . Folgt nun eine Null, geht M zurück in den Zustand q_0 ; falls eine Eins gelesen wird, geht M in den Zustand q_4 .

In den Zuständen q_3 und q_4 entfernt M die restlichen Zeichen vom Band, schreibt eine Null bzw. Eins auf das Band und geht in den Endzustand \bar{q} .

Aufgabe H2 (10 Punkte)

Beschreiben Sie formal eine Turingmaschine, die die Sprache $\{q \in \{0, 1\}^* \mid |w|_0 = |w|_1\}$ entscheidet.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T3

Geben Sie die Gödelnummer $\langle M \rangle$ der unten angegebenen Turingmaschine an.

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, B, q_1, q_3, \delta)$$

δ	0	1	B
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, 0, L)$
q_2	$(q_2, 0, L)$	$(q_2, 1, L)$	(q_3, B, R)

Nutzen Sie dabei die Definition der Gödelnummer aus der Vorlesung.

Lösungsvorschlag

Wir kodieren Zustand q_i als 0^i . Zusätzlich nummerieren wir sowohl das Alphabet durch indem wir $X_1 = 0$, $X_2 = 1$ und $X_3 = B$ setzen, als auch die möglichen Kopfbewegungen indem wir $D_1 = L$, $D_2 = N$ und $D_3 = R$ setzen. Die Gödelnummer ist dann:

$$\begin{array}{cccccccccc}
 111 & 0^1 & 1 & 0^1 & 1 & 0^1 & 1 & 0^1 & 1 & 0^3 \\
 11 & 0^1 & 1 & 0^2 & 1 & 0^1 & 1 & 0^2 & 1 & 0^3 \\
 11 & 0^1 & 1 & 0^3 & 1 & 0^2 & 1 & 0^1 & 1 & 0^1 \\
 11 & 0^2 & 1 & 0^1 & 1 & 0^2 & 1 & 0^1 & 1 & 0^1 \\
 11 & 0^2 & 1 & 0^2 & 1 & 0^2 & 1 & 0^2 & 1 & 0^1 \\
 11 & 0^2 & 1 & 0^3 & 1 & 0^3 & 1 & 0^3 & 1 & 0^3 & 111
 \end{array}$$

Aufgabe T4

Wiederholen Sie kurz das Konzept der universellen Turingmaschine.

Es sei U die universelle Turingmaschine. Wie arbeitet U auf der Eingabe $\langle U \rangle \langle U \rangle \langle U \rangle \langle M \rangle w$? Dabei sind M eine beliebige Turingmaschine und w ein beliebiges Eingabewort.

Lösungsvorschlag

Da die universelle Turingmaschine alle Turingmaschinen simulieren kann, kann sie sich natürlich auch selbst simulieren.

Daher wird U mehrfach geschachtelt simuliert, und der innerste Aufruf simuliert M auf Eingabe w . Das Ergebnis der Berechnung wird dann nach außen „weitergereicht“.

Aufgabe T5

Sind die folgenden Sprachen rekursiv? Begründen Sie Ihre Antwort.

- a) $L_{100} = \{ \langle M \rangle \mid M \text{ h\u00e4lt auf der leeren Eingabe in h\u00f6chstens 100 Schritten} \}$.
- b) $L'_{100} = \{ \langle M \rangle \mid M \text{ besucht h\u00f6chstens 100 Bandpl\u00e4tze auf der leeren Eingabe} \}$.
- a) Die Sprache ist rekursiv. Nach 100 Schritten kann M h\u00f6chstens 100 Bandpl\u00e4tze gelesen haben. Also k\u00f6nnen wir eine TM M_{100} bauen, die L_{100} entscheidet. M_{100} simuliert M auf der leeren Eingabe, und pr\u00fcft ob M nach h\u00f6chstens 100 Schritten h\u00e4lt.
- b) Die Sprache ist rekursiv. Es k\u00f6nnen offensichtlich nur endlich viele Konfigurationen durchlaufen werden die kein Endzustand sind und nur 100 Bandpl\u00e4tze nutzen ($|\Gamma|^{100} \cdot (|Q| - 1) \cdot 100$). Es werden $|\Gamma|^{100} \cdot (|Q| - 1) \cdot 100$ viele Schritte von M auf der leeren Eingabe simuliert, wobei jede Zwischenkonfiguration gespeichert werden kann. Es k\u00f6nnen folgende F\u00e4lle auftreten
- Es werden mehr als 100 Pl\u00e4tze besucht, wir k\u00f6nnen also verwerfen.
 - Eine Konfiguration wird zum zweiten mal besucht, folglich laufen wir in eine Endlosschleife. Da bis jetzt nicht mehr als 100 Pl\u00e4tze besucht wurden, werden auch beim n\u00e4chsten mal nicht mehr besucht, daher k\u00f6nnen wir akzeptieren.
 - Wurde nach $|\Gamma|^{100} \cdot (|Q| - 1) \cdot 100$ Schritten weder eine Konfiguration zweimal besucht, noch gehalten, muss M mehr als 100 Bandpl\u00e4tze besucht haben, wir k\u00f6nnen also verwerfen.

Aufgabe H3 (15 Punkte)

Es ist sehr n\u00fctzlich \u00fcber einen Turingmaschinensimulator zu verf\u00fcgen, wenn man Turingmaschinen entwirft. Ziel dieser Aufgabe ist es, einen solchen Simulator zu implementieren.

Wir wollen genau solche Turingmaschinen simulieren, wie sie in der Vorlesung definiert wurden. Der Simulator soll als Eingabe einen Dateinamen einer Datei erhalten, welche eine Beschreibung der zu simulierenden Turingmaschine $M = \{Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta\}$ enth\u00e4lt, und ein Eingabewort $w \in \Sigma^*$.

Der Simulator soll dann M auf der Eingabe w simulieren und alle durchlaufenen Konfiguration in jeweils einer Zeile ausgeben.

Die Beschreibung einer Turingmaschine in einer Datei ist dabei folgenderma\u00dfen aufgebaut:

1. Die erste Zeile enth\u00e4lt die Anzahl der Zust\u00e4nde n , wobei wir $Q = \{q_1, q_2, \dots, q_n\}$ voraussetzen.
2. Die zweite Zeile enth\u00e4lt alle Zeichen aus Σ .
3. Die dritte Zeile enth\u00e4lt alle Zeichen aus Γ .
4. Die vierte Zeile enth\u00e4lt die Nummer i des Anfangszustands $q_i = q_0$.
5. Die f\u00fcnfte Zeile enth\u00e4lt die Nummer j des Endzustands $q_j = \bar{q}$.

6. Die folgenden Zeilen enthalten je einen Übergang $\delta: (q, a) \mapsto (q', b, D)$ der Übergangsfunktion δ , wobei q, a, q', b und D in dieser Reihenfolge durch jeweils ein Leerzeichen getrennt auftreten.
7. Wir legen fest, daß das Blanksymbol B ist und nicht undefiniert wird.

Folgende Vereinbarungen vereinfachen die Implementierung und Verwendung des Simulators etwas: Sie müssen nicht überprüfen, ob die Beschreibung und das Eingabewort syntaktisch und semantisch korrekt sind. Füttert man den Simulator mit falschen Daten, dann darf etwas beliebiges passieren. Der Simulator muß daher auch nicht alle Zeilen der Eingabe berücksichtigen. Es ist beispielsweise erlaubt, die Zeilen mit Σ und Γ zu ignorieren und nur die Beschreibung von δ zu verwenden. Es ist auch erlaubt, daß δ nicht vollständig angegeben wird und einige Übergänge fehlen. Das macht zum Beispiel dann Sinn, wenn Sie wissen, daß diese Übergänge sowieso nie verwendet werden. So läßt sich viel Schreibarbeit einsparen.

Sie müssen Ihr Programm nicht hocheffizient optimieren, aber es sollte auch nicht zu langsam sein. Längere Simulationen sollten mit nur kurzer Wartezeit durchgeführt werden können. Verwenden Sie eine vernünftige Programmiersprache, die halbwegs bekannt ist und deren Programme von den Tutoren verstanden werden (Java, C, C++, usw. sind alle sehr dafür geeignet. Javascript wäre wohl keine gute Idee).

Zum Schluß läßt sich das gewünschte Verhalten am besten an einem kleinen Beispiel erläutern:

\$ head add.tm	[1]10#1	B11#0[1]B
7	B1[1]0#1	B11#[2]0B
01#	B10[1]#1	B11[2]#1B
01#B	B10#[1]1	B11#[5]1B
1	B10#1[1]	B11#B[5]B
7	B10#[2]1B	B11#[5]BB
1 0 1 0 R	B10#[3]0B	B11[5]#BB
1 1 1 1 R	B10[3]#0B	B1[6]1BBB
1 # 1 # R	B1[4]0#0B	B[6]11BBB
1 B 2 B L	B1[1]1#0B	[6]B11BBB
2 1 3 0 N	B11[1]#0B	BB[7]11BBB
\$./tm add.tm 10#1	B11#[1]0B	\$

Testen Sie Ihren Simulator an verschiedenen kleinen Turingmaschinen.

Erläutern Sie kurz, wie Ihr Simulator funktioniert und geben Sie Protokolle kleiner Beispielläufe und den Quelltext mit ab.

Lösungsvorschlag

Hier ist ein nicht besonders effizientes Programm in der Sprache D, welche das Gewünschte leistet:

```

import std.stdio;
import std.file;
import std.format;
import std.stream;

class TM {
    private int n; // number of states
    private string sigma, gamma; // input and tape alphabets
    private int q0, qbar; // initial and final state
    struct ta {int state; char c; char dir; }
    private ta delta[int][char];

    private int q; // state we are in
    private string left, right; // left and right tape content

    // Have we reached the final state yet?
    bool blocked() {return q == qbar; }

    // Start simulation on word w
    void start(string w) {
        q = q0;
        left = "";
        right = w;
    }

    // Simulate one step
    void step() {
        if(blocked()) return;
        if(right.length == 0) right = "B";
        if(left.length == 0) left = "B";
        int nextstate = delta[q][right[0]].state;
        char symbol_written = delta[q][right[0]].c;
        char dir = delta[q][right[0]].dir;
        q = nextstate;
        right = symbol_written ~ right[1..$];
        if(dir == 'L') {right = left[$ - 1] ~ right; left =
left[0..$ - 1];}
        else if(dir == 'R') {left ~ = right[0]; right =
right[1..$];}
        else assert(dir == 'N');
    }
}

// String representation of current configuration
string configuration() {
    return left ~ '[' ~ std.conv.toString(q) ~ ']' ~ right;
}

// Initialize from a description in a file
this(string filename) {
    assert(filename.isFile());
    Stream file = new BufferedFile(filename);
    char line[] = file.readLine(); // read n
    formattedRead(line, "%d", &n);
    line = file.readLine(); // read Sigma
    line = file.readLine(); // read Gamma
    line = file.readLine(); // read q0
    formattedRead(line, "%d", &q0);
    line = file.readLine(); // read qbar
    formattedRead(line, "%d", &qbar);
    while(!file.eof) {
        line = file.readLine();
        int p, q;
        char char_read, char_write, dir;
        formattedRead(line, "%d %c %d %c %c",
            &p, &char_read, &q, &char_write, &dir);
        delta[p][char_read] = ta(q, char_write, dir);
        assert(dir == 'L' || dir == 'R' || dir == 'N');
    }
}

void main(string args[]) {
    assert(args.length == 3, "Usage: tm M w");
    TM M = new TM(args[1]);
    M.start(args[2]);
    while(!M.blocked()) {
        writeln(M.configuration());
        M.step();
    }
    writeln(M.configuration());
}

```

Aufgabe H4 (8 Punkte)

Entwerfen Sie eine Turingmaschine, welche zwei durch # getrennte, binärkodierte Zahlen entgegennimmt und als Ausgabe ihre binärkodierte Summe präsentiert.

Führen Sie mehrere Simulationen dieser Turingmaschine mithilfe des Simulators aus Aufgabe H3 an aussagekräftigen Beispieleingaben durch.

Lösungsvorschlag

Eine Möglichkeit, die Addition durchzuführen, ist folgende: Es werden abwechselnd die rechte Zahl um eins verringert und die linke um eins erhöht. Das ganze endet, wenn die rechte Zahl 0 war. Dann wird alles bis auf die linke Zahl gelöscht und der Kopf auf das erste Zeichen der linken Zahl gesetzt:

```

7
01#
01#B
1
7
1 0 1 0 R scan right to #
1 1 1 1 R
1 # 1 # R
1 B 2 B L goto rightmost digit of second word
2 1 3 0 N subtract one and goto state 3
2 0 2 1 L
2 # 5 # R if this happens the number was already 0
3 0 3 0 L scan to rightmost digit of first word
3 1 3 1 L
3 # 4 # L
4 0 1 1 N add one to first word and goto 5
4 1 4 0 L
4 B 1 1 N
5 1 5 B R delete second word
5 B 5 B L
5 # 6 B L
6 0 6 0 L return to first word and stop
6 1 6 1 L
6 B 7 B R

```

Zwei typische Simulation mit dieser Turingmaschine erzeugen folgende Ausgabe:

```

$ ./tm add.tm 0#0      B101#[1]11          B11[4]0#01B        B1000#[1]00B
[1]0#0                B101#1[1]1          B11[1]1#01B        B1000#0[1]0B
B0[1]#0               B101#11[1]          B111[1]#01B        B1000#00[1]B
B0#[1]0               B101#1[2]1B         B111#[1]01B        B1000#0[2]0B
B0#0[1]               B101#1[3]0B         B111#0[1]1B        B1000#[2]01B
B0#[2]0B              B101#[3]10B         B111#01[1]B        B1000[2]#11B
B0[2]#1B              B101[3]#10B         B111#0[2]1B        B1000#[5]11B
B0#[5]1B              B10[4]1#10B         B111#0[3]0B        B1000#B[5]1B
B0#B[5]B              B1[4]00#10B         B111#[3]00B        B1000#BB[5]B
B0#[5]BB              B1[1]10#10B         B111[3]#00B        B1000#B[5]BB
B0[5]#BB              B11[1]0#10B         B11[4]1#00B        B1000#[5]BBB
B[6]0BBB              B110[1]#10B         B1[4]10#00B        B1000[5]#BBB
[6]B0BBB              B110#[1]10B         B[4]100#00B        B100[6]0BBBB
BB[7]0BBB             B110#1[1]0B         [4]B000#00B        B10[6]00BBBB
$ ./tm add.tm 101#11  B110#10[1]B         B[1]1000#00B        B1[6]000BBBB
[1]101#11             B110#1[2]0B         B1[1]000#00B        B[6]1000BBBB
B1[1]01#11           B110#[2]11B         B10[1]00#00B        [6]B1000BBBB
B10[1]1#11           B110#[3]01B         B100[1]0#00B        BB[7]1000BBBB
B101[1]#11           B110[3]#01B         B1000[1]#00B        $

```

Natürlich addiert eine solche Turingmaschine relativ langsam, wenn es sich um sehr große Zahlen handelt:

```

$ time ./tm add.tm 1101101011010110#101001110110010 | tail -1
BB[7]111011111010001000BBBBBBBBBBBBBBBB

```

```
real 0m0.451s
user 0m0.444s
sys 0m0.028s
$
```

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T6

1. Zeigen Sie, daß jede RAM durch eine RAM mit einer festen Anzahl von Registern simuliert werden kann.

Hinweis: Als Zwischenschritt können Sie auch eine RAM durch eine Turingmaschine simulieren.

Lösungsvorschlag Wir simulieren erst die RAM mit beliebiger Anzahl der beliebiger Register durch eine TM M , nach VL ist dies möglich. Jetzt muss nur noch gezeigt werden, dass M durch eine RAM mit konstanter Anzahl Register simuliert werden kann. Dazu speichern wir den Zustand in Register 1, dies ist möglich da beliebig große Zahlen in ein Register gespeichert werden können. Den Inhalt des Bandes links von Kopf und rechts von Kopf kann jeweils in Register 2 und 3 gespeichert werden. Der Bandinhalt kann dabei zum Beispiel im Tenärsystem gespeichert werden. Zugriffe auf Bandinhalte kann dann über MOD- und DIV operatoren passieren.

2. Wir wollen zeigen, dass der Befehlssatz der RAM für die Simulation auf die Befehle LOAD, CLOAD, STORE, CADD, CSUB, GOTO, IF $c(0) \neq 0$ GOTO und END eingeschränkt werden kann. Hierzu nutzen wir aus, daß wir nur eine RAM mit konstant vielen Registern simulieren müssen.

Zeigen Sie, wie die Befehle ADD i und INDLOAD i durch den eingeschränkten Befehlssatz ersetzt werden können.

- ADD i

Lösungsvorschlag Der Befehl ADD i bewirkt ein $c(0) = c(0) + c(i)$. Da der Akkumulator an den meisten Befehlen implizit beteiligt ist, und wir das Register i nicht verändern wollen, benutzen wir zwei Hilfsregister $c(k+1) = c(0)$ und $c(k+2) = c(i)$. Etwas abstrahiert und in einer Hochsprache sieht der Befehl dann wie folgt aus.

```
c(k+1)=c(0);  
c(k+2)=c(i);  
while (c(k+2)>0)  
{  
    c(k+1)=c(k+1)+1;  
    c(k+2)=c(k+2)-1;  
}  
c(0)=c(k+1);
```

Dabei wird davon ausgegangen, dass das ursprüngliche RAM-Programm nur die Register $c(0)$ bis $c(k)$ benutzt. Übersetzt lautet der RAM-Befehl dann:

```

STORE k+1
LOAD i
STORE k+2
IF c(0) != 0 GOTO while
GOTO end
$while
LOAD k+1
CADD 1
STORE k+1
LOAD k+2
CSUB 1
STORE k+2
IF c(0) != 0 GOTO while
$end
LOAD k+1

```

- INDLOAD i

Lösungsvorschlag

```

LOAD i;
IF c(0) != 0 GOTO biggerthan0
LOAD 0
GOTO end
$biggerthan0
CSUB 1
IF c(0) != 0 GOTO biggerthan1
LOAD 1
GOTO end
...
$biggerthan(k-2)
CSUB 1
IF c(0) != 0 GOTO biggerthan(k-1)
LOAD k-1
GOTO end
$biggerthan(k-1)
LOAD k
$end

```

Aufgabe T7

Zeigen sie mithilfe der Unterprogrammtechnik, daß die Sprachen

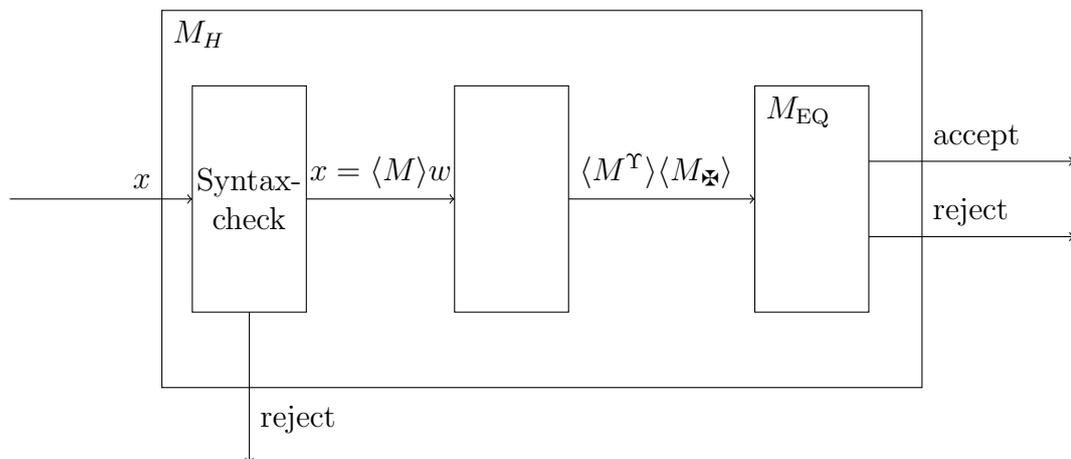
1. $A = \{\langle M \rangle w \mid M \text{ akzeptiert } w\}$.
2. $A_{\text{EQ}} = \{\langle M_1 \rangle \langle M_2 \rangle \mid L(M_1) = L(M_2)\}$.

nicht entscheidbar sind. Verwenden Sie *nicht* den Satz von Rice.

Lösungsvorschlag

1.
 - Zum Widerspruch nehmen wir an, es gibt eine TM M_A , die die Sprache A entscheidet.
 - Gemäß der Definition *rekursiver Sprachen* hält M_A auf jeder Eingabe x und akzeptiert genau dann, wenn $x = \langle M \rangle w \in A$.
 - Wir konstruieren nun eine TM $M_{\bar{D}}$ für das Komplement der Diagonalsprache, die M_A als Unterprogramm verwendet und \bar{D} entscheidet: Zu einer Eingabe x berechnet $M_{\bar{D}}$ zunächst denjenigen Index i , so dass $x = w_i$. Dann berechnet $M_{\bar{D}}$ die Gödelnummer $\langle M_i \rangle$ der Maschine M_i und ruft die Maschine M_A mit der Eingabe $\langle M_i \rangle w_i$ auf. Anschließend übernimmt sie deren Entscheidung.
 - Die TM $M_{\bar{D}}$ entscheidet nun offensichtlich \bar{D} . Ein Widerspruch zur Unentscheidbarkeit von \bar{D} .

2.
 - Zum Widerspruch nehmen wir an, es gibt eine TM M_{EQ} , die die Sprache A_{EQ} entscheidet.
 - Gemäß der Definition *rekursiver Sprachen* hält M_{EQ} auf jeder Eingabe x und akzeptiert genau dann, wenn $x = \langle M_1 \rangle \langle M_2 \rangle \in A_{\text{EQ}}$.
 - Wir konstruieren nun eine TM M_H für das Halteproblem, die M_{EQ} als Unterprogramm verwendet und H entscheidet: Zu einer Eingabe x prüft M_H zunächst, ob $x = \langle M \rangle w$ gilt und die Eingabe somit die richtige Form hat. Ist dies nicht der Fall, so verwirft M_H .
 - Nun berechnet M_H die Gödelnummern zweier Turingmaschinen $\langle M^{\text{r}} \rangle$ und $\langle M_{\text{✕}} \rangle$ mit den nachfolgenden Eigenschaften:
 - M^{r} löscht zunächst ihre Eingabe und schreibt dann w auf das Eingabeband. Dann verhält sich M^{r} genau wie M , bis M hält. Wenn M hält, so akzeptiert M^{r} .
 - $M_{\text{✕}}$ ist eine TM, die auf allen Eingaben hält und die immer akzeptiert.
 - Nun ruft die Turingmaschine M_H die TM M_{EQ} mit der Eingabe $\langle M^{\text{r}} \rangle \langle M_{\text{✕}} \rangle$ auf und übernimmt das Akzeptanzverhalten.



Offensichtlich terminiert die konstruierte TM M_H auf allen Eingaben.

Korrektheit:

Falls w nicht mit einer gültigen Gödelnummer beginnt, so ist $w \notin H$ und M_H verwirft die Eingabe w .

Sei nun $w = \langle M \rangle w$. Es gilt:

$$\begin{aligned} w \in H &\Rightarrow M \text{ hält auf } w \\ &\Rightarrow \langle M^\Upsilon \rangle \text{ akzeptiert } w \\ &\Rightarrow \langle M^\Upsilon \rangle \text{ und } \langle M_{\mathfrak{A}} \rangle \text{ akzeptieren die selben Sprachen} \\ &\Rightarrow \langle M^\Upsilon \rangle \langle M_{\mathfrak{A}} \rangle \in M_{\text{EQ}} \\ &\Rightarrow M_{\text{EQ}} \text{ akzeptiert } \langle M^\Upsilon \rangle \langle M_{\mathfrak{A}} \rangle \\ &\Rightarrow M_H \text{ akzeptiert } w \end{aligned}$$

$$\begin{aligned} w \notin H &\Rightarrow M \text{ hält nicht auf } w \\ &\Rightarrow \langle M^\Upsilon \rangle \text{ hält nicht} \\ &\Rightarrow \langle M^\Upsilon \rangle \text{ und } \langle M_{\mathfrak{A}} \rangle \text{ akzeptieren nicht die selben Sprachen} \\ &\Rightarrow \langle M^\Upsilon \rangle \langle M_{\mathfrak{A}} \rangle \notin M_{\text{EQ}} \\ &\Rightarrow M_{\text{EQ}} \text{ verwirft } \langle M^\Upsilon \rangle \langle M_{\mathfrak{A}} \rangle \\ &\Rightarrow M_H \text{ verwirft } w \end{aligned}$$

Die TM M_H entscheidet damit H . Dies ist aber ein Widerspruch zur Unentscheidbarkeit von H . Die getroffene Annahme, dass die TM M_{EQ} existiert, ist somit falsch und die Sprache A_{EQ} ist unentscheidbar.

Aufgabe H5 (5 Punkte)

Führen Sie den zweiten Teil der Aufgabe T6 für den Befehl `IF c(0) < x GOTO j` durch.

Lösungsvorschlag

Statt nach $c(0) < x$ können wir auch nach $c(0) + 1 - x \leq 0$ fragen. Da x konstant ist, ist diese Umwandlung trivial.

```
STORE k+1
CADD 1
CSUB x
IF c(0)!=0 GOTO continue
LOAD k+1
GOTO j
$continue
LOAD k+1
```

Aufgabe H6 (3 Punkte)

Beweisen oder widerlegen Sie die folgende Behauptung: Eine RAM, deren Registergröße beschränkt ist (zum Beispiel auf 32 bit) kann jede Turingmaschine simulieren.

Lösungsvorschlag Die Aussage ist falsch, da mit endlich großen Registern auch nur endlich viel Speicher adressiert werden kann.

Aufgabe H7 (9 Punkte)

Zeigen Sie mithilfe der Unterprogrammtechnik, daß die Sprache

$$H_1 = \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe und akzeptiert mindestens ein Wort}\}$$

nicht entscheidbar ist. Gehen Sie dabei besonders auf die Korrektheit ein.

Lösungsvorschlag Um $H_{\text{all}} \leq H_1$ zu zeigen, konstruieren wir eine berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$, so dass $x \in H_{\text{all}} \Leftrightarrow f(x) \in H_1$ gilt.

Sei x das Argument von f . Falls x keine gültige Gödelnummer ist, so setze $f(x) = x$. Andernfalls ist $x = \langle M \rangle$ einer TM M . Wir konstruieren nun die Gödelnummer einer TM M' und setzen $f(x) = \langle M' \rangle$. M' arbeitet dabei wie folgt: M' simuliert auf Eingabe w die TM M . Wenn M hält, dann akzeptiert M' .

Offensichtlich ist die Funktion f berechenbar.

Korrektheit:

Falls x keine gültige Gödelnummer ist, so ist $x \notin H_{\text{all}}$ und gemäß der Konstruktion gilt $f(x) \notin H_1$.

Sei nun $x = \langle M \rangle$. Es gilt:

$$\begin{aligned} x \in H_{\text{all}} &\Rightarrow M \text{ hält auf jeder Eingabe} \\ &\Rightarrow M' \text{ hält auf jeder Eingabe und akzeptiert jede Eingabe} \\ &\Rightarrow f(x) = \langle M' \rangle \in H_1 \end{aligned}$$

$$\begin{aligned} x \notin H_{\text{all}} &\Rightarrow M \text{ hält nicht auf jeder Eingabe} \\ &\Rightarrow M' \text{ hält nicht auf jeder Eingabe} \\ &\Rightarrow f(x) = \langle M' \rangle \notin H_1 \end{aligned}$$

Aufgabe H8 (späterer Abgabetermin)

Geben Sie eine Turingmaschine mit fünf Zuständen und Bandalphabet $\Gamma = \{0, 1, B\}$ an, die auf der leeren Eingabe ϵ terminiert, so daß nach Ende der Berechnung eine möglichst hohe Anzahl an 1en auf dem Band steht.

Testen Sie ihre Konstruktion mit Ihrem Simulator vom zweiten Übungsblatt. Geben Sie die Anzahl der 1en an, die nach der Terminierung auf dem Band stehen.

Für diese Aufgabe haben Sie Zeit bis zum ersten Termin ihres Tutoriums nach den Weihnachtsferien. (Sie dürfen Ihre Lösung natürlich auch früher abgeben.)

Lösungsvorschlag TODO

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T8

Betrachten Sie die Sprache $L_{Eigen} = \{ \langle M \rangle \mid M \text{ akzeptiert das Wort } \langle M \rangle \}$. Wiederholen Sie zunächst den Beweis aus der Globalübung, der zeigt, daß L_{Eigen} nicht rekursiv ist. Dieser Beweis benutzte nicht den Satz von Rice—beweisen Sie, daß der Satz von Rice überhaupt nicht auf L_{Eigen} anwendbar ist. Zeigen Sie anschließend, daß L_{Eigen} rekursiv aufzählbar ist.

Lösungsvorschlag

Betrachten wir $\bar{L}_{Eigen} = \{ \langle M \rangle \mid M \text{ akzeptiert das Wort } \langle M \rangle \text{ nicht} \}$. Nehmen wir an, \bar{L}_{Eigen} wäre berechenbar, es existiert also eine TM $M_{\bar{L}_{Eigen}}$, welche \bar{L}_{Eigen} entscheidet. Versuchen wir festzustellen, ob die Gödelnummer von $M_{\bar{L}_{Eigen}}$ in \bar{L}_{Eigen} enthalten ist. Einerseits erhalten wir aus der Definition der Sprache, daß

$$\langle M_{\bar{L}_{Eigen}} \rangle \in \bar{L}_{Eigen} \Leftrightarrow M_{\bar{L}_{Eigen}} \text{ akzeptiert das Wort } \langle M_{\bar{L}_{Eigen}} \rangle \text{ nicht}$$

Andererseits entscheidet $M_{\bar{L}_{Eigen}}$ die Sprache \bar{L}_{Eigen} , und damit gälte, daß

$$\langle M_{\bar{L}_{Eigen}} \rangle \in \bar{L}_{Eigen} \Leftrightarrow M_{\bar{L}_{Eigen}} \text{ akzeptiert das Wort } \langle M_{\bar{L}_{Eigen}} \rangle$$

im Widerspruch zur obigen Feststellung. Damit kann \bar{L}_{Eigen} nicht rekursiv sein und somit folgt, daß auch L_{Eigen} nicht rekursiv ist.

Zeigen wir nun, daß der Satz von Rice hier nicht anwendbar ist.

Wenn wir mithilfe des Satzes von Rice beweisen wollen, daß eine Sprache L nicht rekursiv ist, müssen wir eine Menge von partiellen Funktionen S finden, so daß $L = L(S)$ gilt. Das ist hier aber nicht möglich: Es seien M_1 und M_2 zwei Turingmaschinen, welche beide die Sprache

$$\{ \langle M \rangle \mid M \text{ hat weniger als 100 Zustände} \}$$

erkennen. Dabei soll M_1 weniger und M_2 mehr als 100 Zustände besitzen. Es ist klar, daß es solche Turingmaschinen gibt und beide dieselbe partielle Funktion f berechnen. Wenn $L = L(S)$ ist, dann müssen sowohl M_1 als auch M_2 zu L gehören (wenn $f \in S$) oder beide nicht zu L gehören (wenn $f \notin S$). Wir wissen aber leider, daß $M_1 \in L$ und $M_2 \notin L$.

Aufgabe T9

Es sei

$$L_{Platz} = \{ \langle M \rangle \mid \text{es existiert } w \in \{0, 1\}^*, \text{ so daß } M \text{ auf } w \text{ mehr als } |w| \text{ Platz benötigt} \}.$$

Beweisen Sie, daß L_{Platz} nicht berechenbar ist.

Lösungsvorschlag

Nehmen wir an, daß L_{Platz} berechenbar wäre, es also eine TM $M_{L_{Platz}}$ gibt, die L_{Platz} entscheidet. Wir benutzen $M_{L_{Platz}}$ als Unterprogram, um H_e zu entscheiden. Dazu konstruieren wir eine TM M_{H_e} , die zunächst einen Syntaxcheck der Eingabe durchführt und verwirft, wenn es sich nicht um eine gültige Gödelnummer $\langle M \rangle$ handelt. Anschließend konstruiert sie eine TM M' , welche wie folgt arbeitet: bei einer Eingabe w simuliert M' genau $|w|$ Schritte der Maschine M mit dem leeren Wort als Eingabe—durch eine Vergrößerung des Bandalphabets können wir eine eigene Spur als Arbeitsband von M reservieren. Sollte außerdem M versuchen, in Zellen links der Eingabe zu arbeiten, so verschieben wir den Bandinhalt von M nach rechts. Nach den $|w|$ Schritten sind auf dem Band von M' also auf keinen Fall mehr als $|w|$ Speicherzellen beschrieben. Sollte M innerhalb der simulierten $|w|$ Schritte halten, so beschreiben wir das mehr als $|w|$ Zellen des Bandes von M' .

Wir behaupten, daß $M_{L_{Platz}}$ die Eingabe $\langle M' \rangle$ genau dann akzeptiert, wenn M auf dem leeren Wort hält. TODO

Aufgabe H9 (15 Punkte)

Beweisen oder widerlegen Sie, daß die folgenden Sprachen entscheidbar sind.

1. $L_{q_0} = \{ \langle M \rangle \mid \text{auf keiner Eingabe verläßt } M \text{ ihren Startzustand} \}$

Lösungsvorschlag

Die Sprache ist entscheidbar. Gibt es einen Übergang der Form $(q_0, *) \rightarrow (q_i, x \in \Gamma, *)$, dann ist $\langle M \rangle$ nicht in L_{q_0} , da es eine Eingabe gibt die mit dem Symbol x beginnt und damit verläßt M auf dieser Eingabe den Zustand q_0 schon im ersten Schritt.

2. $L_{Quadrat} = \{ \langle M \rangle \mid M \text{ berechnet die Funktion } n \mapsto n^2 \}$

Lösungsvorschlag

Die Sprache ist nach dem Satz von Rice unentscheidbar.

3. $L_{q_3} = \{ \langle M \rangle \mid M \text{ besucht auf jeder Eingabe Zustand } q_3 \}$

Hinweis: Die Zustände von M sind aufsteigend durchnumeriert. Jede Turingmaschine mit drei oder mehr Zustände besitzt also auch einen Zustand q_3 .

Lösungsvorschlag

Die Sprache ist nicht entscheidbar. Zum Beweis kann z.B. Unterprogrammtechnik verwendet werden. Dabei reicht es aus $\langle M \rangle$ so zu $\langle M' \rangle$ umzuformen, dass in M' alle Übergänge aus q_3 in den Endzustand \bar{q} führen. Außerdem werden alle Übergänge die in M nach \bar{q} gehen so geändert, dass sie in eine Endlosschleife führen.

Nun terminiert M' genau dann wenn M den Zustand q_3 besucht. Folglich könnte mit einer TM die L_{q_3} entscheidet das allgemeine Halteproblem entschieden werden.

4. $L_{1/\sqrt{2}} = \left\{ \langle M \rangle \mid \lim_{n \rightarrow \infty} \frac{|\{ w \mid n = |w|, M \text{ akzeptiert } w \}|}{|\{ w \mid n = |w| \}|} = \frac{1}{\sqrt{2}} \right\}$

Lösungsvorschlag Wir zeigen mit Hilfe des Satzes von Rice, daß $L_{\frac{1}{\sqrt{2}}}$ nicht entscheidbar ist. Dazu betrachten wir die Menge von Funktionen $S = \{f \in \mathcal{R} \mid \lim_{n \rightarrow \infty} \frac{|\{w \mid n=|w|, f(w)=1\}|}{|\{w \mid n=|w|\}|} = \frac{1}{\sqrt{2}}\}$. Offensichtlich ist $L_{\frac{1}{\sqrt{2}}} = L(S)$. Um den Satz von Rice anzuwenden, müssen wir lediglich zeigen, dass $\emptyset \neq S \neq \mathcal{R}$. Die Funktion, die auf jedem Wort eins ist, ist offensichtlich nicht in S enthalten, trivial jedoch in \mathcal{R} . Für den Fall $S \neq \emptyset$ konstruieren wir nun eine Turingmaschine M , die eine Funktion $f_M \in S$ berechnet. Unsere Maschine M akzeptiert schlicht alle Wörter der Form $w = 1^l 0^{l-1}$ mit $l = \lfloor \frac{2^{|w|}}{\sqrt{2}} \rfloor$. Somit ist $f_M \in S$ und damit $S \neq \emptyset$. Durch Anwendung des Satzes von Rice folgt damit die Unentscheidbarkeit von $L_{\frac{1}{\sqrt{2}}}$.

Zum Zeigen der Entscheidbarkeit skizzieren Sie eine Turingmaschine, die die Sprache entscheidet, oder beschreiben Sie ein formales Entscheidungsverfahren. Um Unentscheidbarkeit zu zeigen, nutzen Sie die Unterprogrammtechnik oder den Satz von Rice aus der Vorlesung.

Alle Turingmaschinen, die eine Gödelnummer besitzen, verwenden das Eingabealphabet $\Sigma = \{0, 1\}$ und das Bandalphabet $\Gamma = \{0, 1, B\}$. Ihre Zustände sind $\{q_1, q_2, \dots, q_n\}$.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T10

Es sei $L(M) = \{ w \mid M \text{ akzeptiert } w \}$. Sind die folgenden Sprachen rekursiv, rekursiv aufzählbar oder keins von beidem. Beweisen Sie ihre Aussage.

1. $L_1 = \{ \langle M \rangle \mid L(M) = \emptyset \}$

Lösungsvorschlag

Nach dem Satz von Rice ist L_1 nicht rekursiv. Trivialerweise ist M_{\neg} , die TM die nichts akzeptiert, in der Sprache und M_A , die TM die alles akzeptiert, nicht in der Sprache.

Außerdem ist die Sprache nicht rekursiv aufzählbar, da das Komplement rekursiv aufzählbar ist (vgl. folgende Teilaufgabe). Wären beide Sprachen rekursiv aufzählbar, dann wären auch beide Sprachen rekursiv.

2. $L_2 = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$

Lösungsvorschlag

L_2 ist das Komplement von L_1 , daher kann L_2 nicht rekursiv sein.

Außerdem ist L_2 rekursiv aufzählbar. Hierzu kann ein Aufzähler konstruiert werden, der wie folgt arbeitet: Gehe aller möglichen Eingabewörter $w_i \in \{0,1\}^*$ und Turing Maschinen M_i in kanonischer Reihenfolge durch und simulierte M_i sowie alle Maschinen M_j mit $j < i$ auf w_i und allen Wörtern w_l mit $l < i$ für $|w|$ Schritte. Wird ein Wort w_l von einer TM M_j akzeptiert, dann gebe die Gödelnummer $\langle M_j \rangle$ aus.

3. $L_3 = \{ \langle M \rangle \mid |L(M)| \text{ ist endlich} \}$

Lösungsvorschlag

L_3 ist nach dem Satz von Rice nicht rekursiv. $\langle M_{\neg} \rangle \in L_3$ und $\langle M_A \rangle \notin L_3$.

Außerdem ist L_3 auch nicht rekursiv aufzählbar. Hierzu reduzieren wir \overline{H}_{ϵ} auf L_3 . Hierzu geben wir eine Funktion f an für die gilt: $w \in \overline{H}_{\epsilon} \Leftrightarrow f(w) \in L_3$. Im folgenden sei eine solche Funktion f konstruktiv als TM beschrieben:

- Syntax-Check: Falls Eingabe w keine Gödelnummer ist, dann sei $f(w) = \langle M_{\neg} \rangle$ mit M_{\neg} verwirft alle Wörter.
- Falls $w = \langle M \rangle$ eine Gödelnummer ist, dann sei $f(w) = \langle M' \rangle$ mit M' löscht seine Eingabe und arbeitet auf ϵ . Falls M' hält, dann akzeptiert es immer.

Korrektheit:

$$\begin{aligned}w \in \overline{H}_\epsilon &\Rightarrow w \neq \langle M \rangle \text{ oder } M \text{ h\u00e4lt nicht auf } \epsilon \\ &\Rightarrow M_\neg \text{ akzeptiert kein Wort oder } M \text{ h\u00e4lt nicht auf } \epsilon \\ &\Rightarrow L(M_\neg eg) = \emptyset = L(M') \\ &\Rightarrow f(w) \in L_3 \\ w \notin \overline{H}_\epsilon &\Rightarrow w = \langle M \rangle \\ &\Rightarrow M \text{ h\u00e4lt auf } \epsilon \\ &\Rightarrow M' \text{ akzeptiert alles} \\ &\Rightarrow |L(M')| \text{ nicht endlich} \\ &\Rightarrow f(w) \notin L_3\end{aligned}$$

4. $L_4 = \{ \langle M \rangle \langle M' \rangle \mid L(M) \cap L(M') = \emptyset \}$

L\u00f6sungsvorschlag

L_7 ist nicht rekursiv. Sei M' die TM die alles akzeptiert, dann ist $L(M) \cap L(M') = \emptyset \Leftrightarrow L(M) = \emptyset$. Die Sprache die \u00fcbbrig bleibt $L' = \{ \langle M \rangle \mid L(M) = \emptyset \}$ ist aber nach dem Satz von Rice nicht entscheidbar. Wieder ist $\langle M_\neg \rangle \in L'$ und $\langle M_A \rangle \notin L'$.

Die Sprache ist nicht rekursiv aufz\u00e4hlbar, da wir ihr Komplement $\overline{L_6}$ rekursiv aufz\u00e4hlen k\u00f6nnen. Ein Aufz\u00e4hler f\u00fcr $\overline{L_6}$ w\u00fcrde jeweils abwechselnd ein Wort aus $L(M)$ und $L(M')$ auf jeweils ein Band schreiben und bei jedem Wort \u00fcberpr\u00fcfen ob es schon auf dem anderen Band steht.

Aufgabe T11

Es seien $f, g: \mathbb{Z}^n \rightarrow \mathbb{Z}$ Polynome mit mehreren Variablen.

$$L_{f \leq g} = \{ f, g \mid f \leq g \text{ hat eine ganzzahlige L\u00f6sung} \}$$

Ist $L_{f \leq g}$ entscheidbar? Beweisen sie ihre Aussage.

Hinweis: Eine Beispielinstantz w\u00e4re etwa $x + (x + 7) \cdot y \leq z^7 - xy + 13$.

L\u00f6sungsvorschlag

Das Problem ist nicht entscheidbar. Um das zu zeigen nutzen wir eine Reduktion vom ber\u00fchmten zehnte hilbertsche Problem, welches nach VL unentscheidbar ist. Dazu wandeln wir eine Instanz aus f\u00fcr das hilbertsche Problem in eine Instanz f\u00fcr $L_{f \leq g}$ um. Zu zeigen ist dann, dass die neue Instanz genau eine L\u00f6sung hat, wenn unsere original Instanz eine L\u00f6sung hat.

Wir setzen g auf die Nullfunktion ($g = 0$). Jetzt bleibt zu fragen ob $f \leq 0$ gilt. Sei f die Eingabe f\u00fcr das zehnte hilbertsche Problem. f hat genau dann eine ganzzahlige Nullstelle, wenn $f^2 \leq 0 = g$ eine ganzzahlige L\u00f6sung hat.

Aufgabe H10 (5 Punkte)

Ist das Problem aus Aufgabe T11 rekursiv aufz\u00e4hlbar? Beweisen Sie ihre Aussage.

Lösungsvorschlag

Ja es ist aufzählbar. Es gibt nur endlich viele verschiedene Variablenbelegungen. Diese werden nacheinander durchgegangen und sobald eine erfüllende dabei ist, geben wir diese aus.

Aufgabe H11 (5 Punkte)

Es seien $f, g : \{0, 1\}^* \cup \{\perp\} \rightarrow \{0, 1\}^* \cup \{\perp\}$ partielle Funktionen für die $f(\perp) = \perp$ und $g(\perp) = \perp$ gilt. Beweisen oder widerlegen sie:

Es gibt genau dann eine Turingmaschine, die $f \circ g$ berechnet, wenn es Turingmaschinen M und M' gibt, die f und g berechnen.

Lösungsvorschlag

\Leftarrow : Es werden einfach die beiden TMs nacheinander simuliert.

\Rightarrow : Die andere Richtung muss nicht unbedingt immer gelten. Sei z.B. f die Funktion die alles auf 0 abbildet, diese ist trivialerweise berechenbar. Nun sei g eine Funktion die das Halteproblem lösen möchte, diese können wir aus der TM die $f \circ g$ berechnet allerdings nicht herleiten.

Aufgabe H12 (8 Punkte)

Ist die Sprache $L = \{ \langle M \rangle \langle M' \rangle \mid L(M) \subseteq L(M') \}$ rekursiv aufzählbar? Beweisen Sie ihre Aussage.

Lösungsvorschlag

Die Sprache ist nicht rekursiv aufzählbar. Wir definieren die Funktion f , die die Codierung einer Turingmaschine $\langle M \rangle$ auf die Codierung zweier Maschinen $\langle M_a \rangle \langle M \rangle^*$ abbildet, wobei M_a eine feste Turingmaschine sei, die alle Eingaben akzeptiert und M^* eine Turingmaschine, die M simuliert und eine Eingabe w akzeptiert wenn M auf w hält.

Weil $L(M_a) = \{0, 1\}^*$ ist, gilt folgendes:

$$\langle M \rangle \in H_{All} \Leftrightarrow f(\langle M \rangle) \in L_{H12}$$

Daher können wir mit einem hypothetischen Aufzähler $M_{\#L}$ für L einen Aufzähler $M_{\#H_{All}}$ konstruieren: wir gönnen uns eine Turingmaschine mit drei Bändern und simulieren $M_{\#L}$ auf dem ersten Band. Die Ausgabe von $M_{\#L}$ geben wir dabei auf das zweite Band, dort erscheinen also nacheinander alle Wörter aus L . Gleichzeitig (d.h. im Wechselschritt mit der Simulation von $M_{\#L}$) zählen wir auf dem dritten Band Turingmaschinen auf und schreiben für jede Maschine M die Codierung $\langle M \rangle \# f(M) \#\#$, also $\langle M \rangle \# \langle M_a \rangle \langle M^* \rangle \#\#$ auf dieses Band. Nun wird unser simulierter Aufzähler irgendwann den String $\langle M_a \rangle \langle M^* \rangle$ ausgeben, sollte M tatsächlich auf allen Eingaben halten. Passiert dies, so löschen wir $\langle M_a \rangle \langle M^* \rangle$ vom zweiten und $\langle M \rangle \# \langle M_a \rangle \langle M^* \rangle \#\#$ vom dritten Band und geben M aus.

Diese Konstruktion wird jede Maschine M , die auf allen Eingaben hält, nach endlicher Zeit ausgeben und ist damit ein Aufzähler für H_{All} . Wir schließen daraus, dass es keinen Aufzähler für L geben kann.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T12

Sind die folgenden Instanzen des Postschen Korrespondenzproblems lösbar? Finden Sie eine Lösung oder zeigen Sie die Unlösbarkeit!

a)

10
1

,

11
01

,

01
0

b)

10
1

,

11
01

,

01
0

,

0
0100

c)

1
0

,

0000
0

,

0
01

Lösungsvorschlag.

a: Oben länger oder gleich wie unten, bei Anfangskarte länger.

b: Lösung 3, 1, 1, 4.

c: Erste und letzte Karte müssen gleich oft vorkommen, dann sind durch diese Karten aber schon mehr Nullen unten als Einsen. Das kann man durch die mittlere Karte nur noch schlimmer machen.

Aufgabe T13

Wenn das Postsche Korrespondenzproblem so modifiziert wird, daß nur ein unäres Alphabet verwendet wird, ist es dann immer noch unentscheidbar?

Lösungsvorschlag.

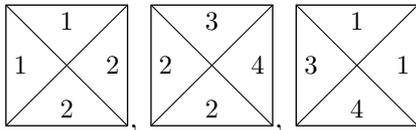
Nein, es wird entscheidbar. Es hat nämlich genau dann eine Lösung, wenn es sowohl eine Karte (u_i, v_i) mit $|u_i| \geq |v_i|$ gibt, als auch eine Karte (u_j, v_j) mit $|u_j| \leq |v_j|$. Dann kann man nämlich einfach $|u_i| - |v_i|$ mal die j te Karte nehmen und einfach $|v_j| - |u_j|$ mal die i te Karte.

Andererseits gibt es sonst sicher keine Lösung, da obere Wörter ja immer länger als untere Wörter sind (oder umgekehrt).

Aufgabe T14

In dieser Aufgabe betrachten wir das folgende Puzzleproblem: Gegeben ist eine Menge von quadratischen Steinen, deren vier Kanten jeweils mit natürlichen Zahlen dekoriert sind. Gesucht ist eine Strategie, eine unendliche Ebene vollständig mit Kopien dieser Steine zu bedecken, wobei aneinanderliegende Kanten stets dieselben Zahlen tragen sollen (es handelt sich gewissermaßen um die unendliche Version des Spiels Tetravex, welches aber, historisch gesehen, selbst als endliche Version unseres Problems entstand).

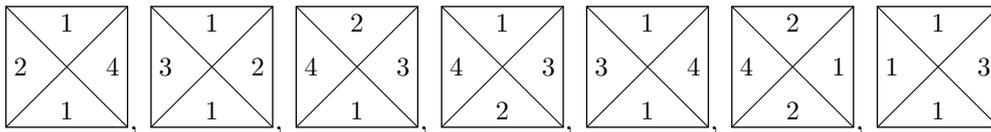
1. Ist diese Instanz lösbar?



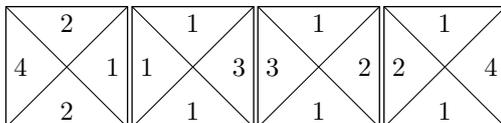
Lösungsvorschlag. Nein: Da oben stets ungerade und unten stets gerade Zahlen auf den Steinen stehen, lassen sich keine zwei Steine übereinanderlegen, also erst recht nicht unendlich viele.

Wer mehr über dieses Problem wissen will, möge im Buch *The Art of Computer Programming, Volume 1: Fundamental Algorithms* von Donald Knuth den Abschnitt 2.3.4.3 lesen.

2. Ist diese Instanz lösbar?



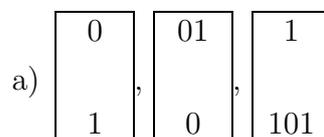
Lösungsvorschlag. Ja: das folgende Rechteck kann unendlich oft aneinandergelagt werden.



Aufgabe H13 (16 Punkte)

Schreiben Sie ein Programm, das eine Instanz des Postschen Korrespondenzproblems über dem Alphabet $\{0, 1\}^*$ einlesen kann und anschließend zu lösen versucht. Wir empfehlen, Backtracking mit einem Abbruch bei einer vorgegebenen Suchtiefe zu verwenden. Das Programm soll entweder eine gefundene Lösung ausgeben, sagen, daß es keine Lösung gibt, oder zugeben, daß es unfähig war diese Instanz zu lösen.

Geben Sie einen Ausdruck Ihres Programms ab und Protokolle über die Arbeit Ihres Programms auf den Instanzen aus Aufgabe T12 und den folgenden:



$$b) \begin{array}{|c|} \hline 01 \\ \hline 000 \\ \hline \end{array}, \begin{array}{|c|} \hline 10 \\ \hline 111 \\ \hline \end{array}, \begin{array}{|c|} \hline 0 \\ \hline 1 \\ \hline \end{array}, \begin{array}{|c|} \hline 0 \\ \hline 010 \\ \hline \end{array}, \begin{array}{|c|} \hline 001 \\ \hline 01 \\ \hline \end{array}, \begin{array}{|c|} \hline 10 \\ \hline 1 \\ \hline \end{array}, \begin{array}{|c|} \hline 010 \\ \hline 11 \\ \hline \end{array}$$

Bedenken Sie, daß die Tutoren in der Lage sein müssen, Ihr Programm nachzuvollziehen.

Lösungsvorschlag. Bei der oberen gibt es eine Lösung der Länge 44 (nämlich 2 1 2 2 1 2 1 2 2 1 2 2 3 2 2 3 3 2 1 1 3 2 2 1 2 3 2 2 1 1 2 3 2 2 2 1 3 2 3 2 3 3 1 3), bei der zweiten gar keine.

Lassen Sie Ihr Programm *nicht* auf diesen Instanzen laufen:

$$c) \begin{array}{|c|} \hline 1101 \\ \hline 1 \\ \hline \end{array}, \begin{array}{|c|} \hline 0110 \\ \hline 11 \\ \hline \end{array}, \begin{array}{|c|} \hline 1 \\ \hline 110 \\ \hline \end{array}$$

$$d) \begin{array}{|c|} \hline 10 \\ \hline 0 \\ \hline \end{array}, \begin{array}{|c|} \hline 0 \\ \hline 001 \\ \hline \end{array}, \begin{array}{|c|} \hline 100 \\ \hline 1 \\ \hline \end{array}$$

$$e) \begin{array}{|c|} \hline 101 \\ \hline 0 \\ \hline \end{array}, \begin{array}{|c|} \hline 1 \\ \hline 101 \\ \hline \end{array}, \begin{array}{|c|} \hline 0 \\ \hline 1 \\ \hline \end{array}$$

Aufgabe H14 (6 Punkte)

Ist folgende Variante des Postschen Korrespondenzproblems entscheidbar? Wir verlangen, daß das Alphabet $\{0,1\}^*$ ist und jedes Wort auf jeder Karte höchstens die Länge 17 besitzt.

Lösungsvorschlag.

Da wir doppelt vorkommende Karten aus einer PKP-Instanz entfernen können gibt es nur endlich viele Instanzen, welche die obige Bedingung erfüllen: es gibt nur 2^{34} verschiedene Karten und daher höchstens $2^{2^{34}}$ Instanzen.

Damit ist diese Variante des PKP entscheidbar, denn die Sprache ist endlich. Dies erscheint zunächst kontraintuitiv, man bedenke aber, dass Entscheidbarkeit lediglich die *Existenz* einer TM voraussetzt, welche die Sprache entscheidet, nicht die *Konstruierbarkeit* dieser TM.

Betrachten wir einfach alle TMs, die als Eingabe die obigen PKP-Instanzen bekommen, und für jede gültige Eingabe eine 0 oder 1 ausgeben (und ansonsten etwas anderes), so entscheidet eine dieser Maschinen die obige Sprache. Dazu muss diese Maschine noch nicht einmal die obigen Instanzen lösen, stattdessen könnte die (richtige) Antwort schlicht in den Zuständen der TM kodiert sein.

Aufgabe H15 (Bonusaufgabe, sehr viele Punkte)

Ist das Puzzleproblem aus Aufgabe T14 entscheidbar?

Hinweis: Diese Aufgabe ist sehr schwierig.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T15

Entwickeln Sie ein WHILE-Programm, daß den Wert 2^{x_1} berechnet. Analysieren Sie die Laufzeit ihres Programmes im uniformen und im logarithmischen Kostenmaß.

Lösungsvorschlag.

Unter der Annahme, daß alle Variablen die nicht zur Eingabe gehören mit 0 initialisiert sind, berechnet das folgende Programm 2^{x_1} :

Eingabe: $x_1 \in \mathbb{N}$

```
 $x_2 := x_2 + 1;$   
WHILE  $x_1 \neq 0$  DO  
   $x_3 := x_2 + 0;$   
  WHILE  $x_3 \neq 0$  DO  
     $x_2 := x_2 + 1;$   
     $x_3 := x_3 - 1$   
  END;  
   $x_1 := x_1 - 1$   
END;  
 $x_0 := x_2 + 0$ 
```

Ausgabe: x_0

Das aktuelle Zwischenergebnis steht immer in x_2 während x_1 und x_3 als Schleifenvariablen genutzt werden.

Im uniformen Kostenmaß beträgt die Laufzeit $O(2^n)$. Im logarithmischen Kostenmaß kommt ein Faktor $n = \log(2^n)$ hinzu, da die Additionen $x = x + 1$ hier $\log(x)$ Zeit benötigen.

Aufgabe T16

Die Ackermannfunktion $A : \mathbb{N}^2 \rightarrow \mathbb{N}$ wurde in der Vorlesung folgendermaßen definiert:

$$\begin{aligned} A(0, m) &= m + 1 && \text{für } m \geq 0 \\ A(n + 1, 0) &= A(n, 1) && \text{für } n \geq 0 \\ A(n + 1, m + 1) &= A(n, A(n + 1, m)) && \text{für } n, m \geq 0 \end{aligned}$$

- Zeigen Sie, daß die Ackermannfunktion für alle Parameter $n, m \in \mathbb{N}$ terminiert.
- Beweisen Sie durch Induktion nach n folgende Aussage:

$$A(n, m) \leq A(n + 1, m - 1) \text{ für alle } m \geq 0$$

Hinweis: Nutzen Sie die Monotonie der Ackermannfunktion in beiden Parametern aus.

Lösungsvorschlag.

- a) In jedem Schritt wird entweder m verringert oder m erhöht und n verringert. Jedes Mal wenn m Null erreicht, wird n verringert, also muss auch n irgendwann Null erreichen. Man beachte allerdings daß bei Verringerung von n keine obere Schranke für das Wachstum von m in den Funktionsaufrufen gibt.
- b) Induktionsanfang: Nach Definition gilt $A(n + 1, 0) := A(n, 1)$ also insbesondere $A(n, 1) \leq A(n + 1, 0)$.

Induktionsschritt: Sei die Behauptung jetzt schon für $n \geq 1$ bewiesen.

Aufgrund der Definition $A(0, m) := m + 1$ und der Monotonie im ersten Parameter gilt $m + 1 \leq A(n, m)$ für alle $m \geq 0$.

Damit gilt $A(n, m + 1) \leq A(n, A(n, m)) \leq A(n, A(n + 1, m - 1)) = A(n + 1, m)$, was zu zeigen war. Dabei geht in die erste Ungleichung die Monotonie im zweiten Parameter ein, in die zweite die Induktionsvoraussetzung, und in die letzte Gleichung die Definition.

Aufgabe T17

Sind WHILE-Programme immer noch Turing-mächtig, wenn die Zuweisungen $x_i := x_j + c$ nur noch für $c \in \{-1, 0\}$ erlaubt sind?

Lösungsvorschlag.

Wir zeigen für folgende (WHILE-berechenbare) Funktion, daß sie nicht durch die eingeschränkten WHILE-Programme berechenbar ist: $f : \mathbb{N} \rightarrow \mathbb{N}$, $f(n) = 1$ für alle $n \in \mathbb{N}$. Dies geht sehr einfach, in dem man per Induktion zeigt: Wenn alle Variablen zu Beginn eines WHILE-Programms mit 0 belegt sind, gilt dies auch nach der Ausführung.

Induktionsanfang: Das WHILE-Programm hat die Form $x_i := x_j + c$ für $c \in \{-1, 0\}$. Hier gilt die Aussage offensichtlich.

Induktionsschluss:

Fall 1: Das WHILE-Programm hat die Form WHILE $x_i \neq 0$ DO P END. P wird offensichtlich nicht ausgeführt, also ändert sich die Belegung der Variablen nicht.

Fall 2: Das WHILE-Programm hat die Form $P_1; P_2$. Weder P_1 noch P_2 ändert die Belegung der Variablen, also gilt auch hier die Aussage

Somit können mit den eingeschränkten WHILE-Programmen höchstens Funktionen mit $f(0) = 0$ berechnet werden.

Aufgabe H16 (8 Punkte)

Betrachten Sie die nachfolgenden Varianten des Euklidischen Algorithmus (entnommen aus Wikipedia) zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen a und b .

Variante 1:

Eingabe: $a, b \in \mathbb{N}$
 While $a \neq b$
 If $a > b$
 then $a := a - b$
 else $b := b - a$
 End While
Ausgabe: a

Variante 2:

Eingabe: $a, b \in \mathbb{N}$
 While $b > 0$
 $r := a \bmod b$
 $a := b$
 $b := r$
 End While
Ausgabe: a

Kommentar

Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler von zwei natürlichen Zahlen a und b . Die in der Aufgabenstellung angegebenen Varianten beruhen auf dem folgenden Lemma.

Lemma 1 Falls $b \neq 0$, dann gilt

$$\begin{aligned} \text{ggT}(a, b) &= \text{ggT}(a - b, b) \\ \text{ggT}(a, b) &= \text{ggT}(a \bmod b, b) . \end{aligned}$$

Wir werden im Folgenden o.B.d.A. annehmen, daß $a \geq b$ gilt.

1. Bestimmen Sie eine möglichst scharfe untere Schranke für die Worst-Case-Laufzeit von Variante 1 im uniformen Kostenmaß.

Lösungsvorschlag

Betrachte den Fall $b = 1$. Wie oben angenommen gilt $a \geq b$. Da entweder $a = b$ oder $a > b$ gilt, wird in jedem Durchlauf der WHILE-Schleife die Anweisung $a := a - b$ ausgeführt. Das heißt aber, daß in jeder Iteration Eins von a subtrahiert wird und somit die WHILE-Schleife $\Omega(a)$ mal ausgeführt wird. Folglich ist die Worst-Case-Laufzeit von Variante 1 durch $\Omega(a)$ nach unten beschränkt.

2. Bestimmen Sie eine möglichst scharfe obere Schranke für die Worst-Case-Laufzeit von Variante 2 im uniformen Kostenmaß.

Lösungsvorschlag

Variante 2 berechnet in jeder Iteration der WHILE-Schleife zunächst $r = a \bmod b$ und setzt dann $a := b$ und $b := r$. Da wie oben angenommen $a \geq b$ gilt, lässt sich der Wert von r nach der ersten Runde durch $r = a \bmod b \leq \frac{a}{2}$ abschätzen (betrachte hierzu den Fall $b = \lfloor a/2 \rfloor + 1$). Somit sind nach zwei Runden sowohl a als auch b nur noch höchstens halb so groß wie die größere der beiden Eingaben, d.h. hier a . Nach $2 \cdot k$ Runden gilt also $a, b \leq \frac{a}{2^k}$. Die Anzahl der Iterationen beträgt somit maximal $2 \cdot \log_2(a)$. Damit berechnet Variante 2 den größten gemeinsamen Teiler von a und b mit höchstens $O(\log(a))$ vielen Modulo-Operationen. Folglich ist die Worst-Case-Laufzeit von Variante 2 durch $O(\log(a))$ nach oben beschränkt.

3. Nutzen Sie Ihre Abschätzungen, um zu zeigen, daß sich die uniformen Worst-Case-Laufzeiten beider Varianten durch einen exponentiellen Faktor unterscheiden.

Lösungsvorschlag

Wie in den Aufgabenteilen oben gezeigt, hat Variante 1 eine Worst-Case-Laufzeit von $\Omega(a)$, Variante 2 hingegen eine von $O(\log(a))$. Die Eingabelängen von a und b können durch $\log(a)$ bzw. $\log(b)$ abgeschätzt werden. Da wir angenommen haben, daß $a \geq b$ gilt, ist die Eingabelänge folglich durch $2 \cdot \log(a)$ beschränkt. Im Vergleich zur Eingabelänge benötigt Variante 2 also polynomielle und Variante 1 exponentielle Zeit.

4. Stimmt diese Aussage auch bezüglich der Laufzeiten im logarithmischen Kostenmaß?

Lösungsvorschlag

Ja, die Aussage stimmt weiterhin. Der Zeitaufwand im logarithmischen Kostenmaß zur Berechnung von einer Zuweisung, eines Vergleichs oder einer Subtraktion zweier Zahlen $a, b \in \mathbb{N}$ mit $a \geq b$ ist $O(\log(a))$. Die Berechnung von $(a \bmod b)$ kann mit Hilfe einer Division, einer Multiplikation und einer Subtraktion durchgeführt werden. Eine Multiplikation und eine Division benötigen nach der Schulmethode $O(\log^2(a))$ viele Schritte. Demnach wird der Aufwand von Variante 2 im Vergleich zum uniformen Kostenmaß also höchstens um einen polylogarithmischen Faktor vergrößert. Der exponentielle Gap bleibt also bestehen.

Aufgabe H17 (6 Punkte)

Geben Sie ein LOOP-Programm an, daß folgende Sprache akzeptiert:

$$L = \{w \in \{0, 1\}^* \mid |w|_0 = |w|_1\}$$

wobei $|w|_i$ die Anzahl der Stellen in w angibt, an denen die Ziffer i steht.

Gehen Sie davon aus, daß die Eingabe in der Variable x_1 als Binärzahl kodiert steht und das zudem die Länge der Eingabe in der Variable x_2 zu finden ist. Wenn der Wert x_1 in der Sprache L enthalten ist, soll am Ende des Programs x_0 eine Eins enthalten, ansonsten Null.

Nehmen Sie an, daß die Subtraktion von Variablen mit dem Wert 0 wiederum 0 ergibt (Variablen können also nie negative Werte enthalten). Zudem setzen wir voraus, daß die Eingabe x_1 nie das leere Wort enthält.

Lösungsvorschlag.

Eingabe: $x_1 \in \mathbb{N}$, $x_2 \in \mathbb{N}$

$x_6 := 0$; // Anzahl Nullen

$x_7 := 0$; // Anzahl Einzen

LOOP x_2 DO

$x_4 := 1$;

$x_5 := 0$;

$x_3 := x_1$;

 LOOP x_3 DO

$x_{10} := 1$;

$x_{11} := x_4$;

 LOOP x_{11} DO

$x_{10} := 0$;

$x_4 := 0$;

$x_5 := 1$;

$x_1 := x_1 - 1$;

 END

 LOOP x_{10} DO

$x_{11} := x_5$;

 LOOP x_{11} DO

$x_4 := 1$;

$x_5 := 0$;

 END

 END

END

LOOP x_4 DO // Gerade

$x_6 := x_6 + 1$;

END

LOOP x_5 DO // Ungerade

$x_7 := x_7 + 1$;

END

END

$x_8 := x_6$;

$x_9 := x_7$;

LOOP x_7 DO

$x_8 := x_8 - 1$;

END

LOOP x_6 DO

$x_9 := x_9 - 1$;

END

$x_0 := 1$;

LOOP x_8 DO

$x_0 := 0$;

END

LOOP x_9 DO

$x_0 := 0$;

END

Ausgabe: x_0

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T18

- a) Wiederholen Sie die Definition der O -Notation.
b) Welche der folgenden Aussagen sind wahr?

1. $n^2 = O(2^n)$
2. $\sqrt{n} = O(n/\log n)$
3. $\sqrt{n-1} = \sqrt{n} + O(1)$
4. $\int_0^{O(n)} O(t) dt = O(n^2)$
5. $(\log n)^n = O(n^{\log n})$

Lösungsvorschlag.

1. $n^2 = O(2^n)$
Wahr.
2. $\sqrt{n} = O\left(\frac{n}{\log n}\right)$
Wahr.
3. $\sqrt{n-1} = \sqrt{n} + O(1)$
Wahr.
4. $\int_0^{O(n)} O(t) dt = O(n^2)$
Wahr.
5. $(\log n)^n = O(n^{\log n})$
Falsch.

Aufgabe T19

Wir bezeichnen mit p_N die N -te Primzahl.

Zu gegebener Eingabe $N \in \mathbb{N}$ sei die kleinste Primzahl gesucht, welche größer als N ist.
Ist diese Aufgabe in polynomieller Zeit lösbar, falls

- a) N in unärer Kodierung vorliegt,
- b) N in binärer Kodierung vorliegt?

Hinweis zu Teilaufgabe b: Cramérs Vermutung von 1936 besagt, daß die Primzahllücke $g(p_n) = p_{n+1} - p_n$ zwischen der n -ten und $(n + 1)$ -ten Primzahl durch

$$g(p_n) = O((\ln p_n)^2)$$

beschränkt ist.

Lösungsvorschlag.

Zunächst erinnern wir daran, daß Zahlen auf Primalität zu testen in polynomieller Zeit möglich ist.

- a) In der unären Kodierung reicht es auszunutzen dass zwischen N und $2N$ immer eine Primzahl zu finden ist. Die $(N+1)$ -te Primzahl befindet sich also in diesem Intervall. Testen wir jede Zahl dazwischen, finden wir nach höchstens $O(N)$ Schritten die nächste Primzahl. Damit ergibt sich eine Laufzeit von $O(N \cdot P)$, wobei P die Laufzeit des Primzahltestes darstellen soll.
- b) Wendet man hier die Lösung aus Aufgabenteil a) an, ergibt sich eine exponentielle Laufzeit. Dies liegt daran, dass wir N Zahlen testen müssen, was im logarithmischen Kostenmaß exponentiell in der Eingabelänge ist. Nutzen wir jedoch Cramérs Vermutung, ergibt sich das nur logarithmisch viele Zahlen getestet werden müssen. Was wieder zu einer polynomieller Laufzeit führt. Damit erhalten wir einen Algorithmus, der entweder in polynomieller Zeit läuft oder aber eine wichtige mathematische Vermutung widerlegt.

Aufgabe H18 (5 Punkte)

Für einen beliebigen Algorithmus sei $t(n) \geq n$ die Laufzeit im uniformen Kostenmaß und $t'(n)$ die Laufzeit im logarithmischen Kostenmaß.

Finden Sie eine möglichst interessante Beziehung zwischen $t(n)$ und $t'(n)$.

Lösungsvorschlag.

Ein einfaches Beispiel für einen Algorithmus der im uniformen Kostenmaß Laufzeit $O(n)$ hat und im logarithmischen Kostenmaß $O(2^n)$ ist der folgende:

Eingabe: $n \in \mathbb{N}$

$x = 2$

For $i = 1, \dots, n$ Do

$x = x^2$

End For

Ausgabe: x

Hier wird $x = 2^{2^n}$ berechnet was als Ausgabe für sich genommen schon einen Platzbedarf von 2^n hat. Im uniformen Kostenmaß wird die Schleife allerdings nur n mal gezählt und die Ausgabe in einem Schritt ausgegeben. Das logarithmische Kostenmaß ist hier viel realistischer, da hier jeder Schleifendurchlauf mit bis zu 2^n eingeht.

Aufgabe H19 (10 Punkte)

Welche Funktion berechnet der folgende Algorithmus?

Analysieren Sie seine Laufzeit sowohl im uniformen als auch im logarithmischen Kostenmaß.

Eingabe: $a \in \mathbb{N}$

$b = 2$

while $a \neq 0$

$b = b \cdot b$

$a = a - 1$

end while

$c = 2$

while $b \neq 0$

$c = c \cdot c$

$b = b - 1$

end while

Ausgabe: c

Lösungsvorschlag.

Nach der ersten Schleife hat b den Wert $b = 2^{2^a}$. Nach Durchlauf der zweiten Schleife ist, analog, $c = 2^{2^b} = 2^{2^{2^a}}$.

uniform: Unter der Voraussetzung das multiplizieren lediglich eine Zeiteinheit kostet, wird die Laufzeit des Algorithmus durch die zweite Schleife dominiert: insgesamt benötigt er $O(2^{2^a})$ Operationen.

log.: Im letzten Durchlauf der zweiten Schleife wird die Zahl $2^{2^{b-1}}$ mit sich selbst multipliziert, diese Operation dominiert die Laufzeit des Algorithmus: im logarithmischen Kostenmaß kostet sie nämlich $\log 2^{2^{b-1}} = 2^{2^{a-1}}$ Zeit. Damit ist die Laufzeit des obigen Algorithmus $O(2^{2^a})$.

Aufgabe H20 (8 Punkte)

Als Eingabe sei eine Menge von Intervallen

$$\left\{ \left[\frac{a_1}{b_1}, \frac{c_1}{d_1} \right], \left[\frac{a_2}{b_2}, \frac{c_2}{d_2} \right], \left[\frac{a_3}{b_3}, \frac{c_3}{d_3} \right], \dots, \left[\frac{a_n}{b_n}, \frac{c_n}{d_n} \right] \right\}$$

gegeben.

Die Anfangs- und Endpunkte jedes Intervalls sind rationale Zahlen, welche durch die Binärdarstellung ihres Zählers und Nenners kodiert werden.

Ist in polynomieller Zeit berechenbar, ob der Schnitt zwischen allen gegebenen Intervallen leer ist?

Falls Sie der Meinung sind, daß die Antwort *ja* ist, dann geben Sie einen Algorithmus an, der die Aufgabe in polynomieller Zeit löst. Wie groß ist die Laufzeit Ihres Algorithmus?

Lösungsvorschlag.

Gehe die Intervalle durch und merke das Maximum über die Anfangspunkte a sowie das Minimum über die Endpunkte e . Falls $a \leq e$ dann ist der Schnitt nicht leer. Jetzt liegt nämlich genau das Intervall $[a, e]$ in allen gegebenen Intervallen. Ansonsten ist der Schnitt leer.

Die Laufzeit im logarithmischen Kostenmaß ist in $O(n \log n)$, da Vergleiche zwischen zwei Intervallgrenzen in $O(\log n)$ machbar sind und alle Intervalle nur einmal aufgezählt werden.

Aufgabe H21 (Bonusaufgabe)

Gegeben sind zwei Mengen von natürlichen Zahlen $\{k_1, \dots, k_n\}$ und $\{l_1, \dots, l_n\}$, die jeweils als durch Trennzeichen separierte Binärdarstellungen als Eingabestring repräsentiert werden.

Kann die folgende Frage in polynomieller Zeit beantwortet werden?

$$\sqrt{k_1} + \dots + \sqrt{k_n} < \sqrt{l_1} + \dots + \sqrt{l_n}$$

Hinweis: Vorsicht! Dies ist eine seit vielen Jahren offene Frage, deren Beantwortung weitreichende Konsequenzen hätte. Es ist heute noch nicht einmal bekannt, ob dieses Problem in NP liegt.

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T20

Beweisen Sie die folgende Aussage: Die Entscheidungsvariante von BIN PACKING ist genau dann in polynomieller Zeit lösbar, wenn dies auch für die Optimierungsvariante gilt.

Lösungsvorschlag.

Wir nehmen also an, dass die Entscheidungsvariante des BPP in P ist und konstruieren einen Polynomialzeit-Algorithmus \mathcal{A}_{OPT} , der eine optimale Lösung des BPP berechnet, und dabei den Polynomialzeit-Algorithmus für die Entscheidungsvariante \mathcal{A}_{ENT} als Unterprogramm benutzt.

Wir nutzen aus, dass man zwei Objekte x und y zusammenkleben kann, indem man beide löscht und durch ein neues Objekt x' mit Gewicht $w(x') = w(x) + w(y)$ ersetzt. Damit wird erzwungen, dass x und y im gleichen Behälter landen. Falls sich dadurch der Wert der Lösung nicht ändert, so gibt es eine Verteilung, die bei gleichbleibender Qualität x und y dem gleichen Behälter zuordnet. Beachte, dass die Lösung durch das Zusammenkleben nicht besser werden kann.

Im Folgenden nehmen wir an, dass der Wert b_{opt} der optimalen Lösung durch ein Orakel gegeben ist. Wir können dieses Orakel später durch eine binärer Suche ersetzen und damit b_{opt} in Polynomialzeit finden. Der Algorithmus \mathcal{A}_{OPT} zur Berechnung der optimalen Verteilung geht nun wie folgt vor:

- \mathcal{A}_{OPT} iteriert über alle Paare (x, y) von Objekten.
- Dabei klebt \mathcal{A}_{OPT} das Paar (x, y) zusammen und prüft mit Hilfe von \mathcal{A}_{ENT} , ob es eine Verteilung der Objekte auf b_{opt} Behälter gibt.
 - Falls ja, bleiben x und y zusammengeklebt und der Algorithmus wird rekursiv mit dem zusammengeklebten Objekt fortgesetzt bis die Anzahl der Objekte b_{opt} ist.
 - Falls nein, dürfen x und y nicht zusammengeklebt werden und es wird das nächste Paar gewählt.

Die auf diese Art erhaltene Verteilung ordnet jedem Behälter ein zusammengeklebtes Objekt zu. Zerlegt man diese Objekte wieder in seine Bestandteile, d.h. die einzelnen Objekte, aus denen es zusammengeklebt worden ist, erhält man die genaue Verteilung.

In jedem Rekursionsschritt wird die Eingabe um ein Objekt verringert; also werden, wenn n die Anzahl der ursprünglichen Objekte ist, $n - b_{\text{opt}}$ Rekursionen durchgeführt. Dabei benötigt jeder Rekursionsschritt maximal $n \cdot (n - 1)$ viele Aufrufe von \mathcal{A} . Folglich ist die Laufzeit von \mathcal{A}_{OPT} polynomiell in der Eingabelänge beschränkt.

Aufgabe T21

Wir betrachten folgendes Problem: Als Eingabe sind ganzzahlige kartesische Koordinaten von n Punkten in der Ebene sowie weitere Zahlen k und A gegeben. Die Frage ist, ob es k Kreise in der Ebene gibt, welche zusammen alle n Punkte umfassen und deren Flächensumme höchstens A ist.

Beschreiben Sie, wie eine nichtdeterministische Turingmaschine dieses Problem in polynomialer Zeit lösen kann?

Aufgabe T22

Wir betrachten das Problem SORTING BY REVERSALS. Eine gegebene Permutation π soll durch höchstens k viele Vertauschungen (*reversals*) ρ in eine zweite gegebene Permutation σ verwandelt werden. Ein *reversal* $\rho(i, j)$ vertauscht dabei die Reihenfolge aller Elemente im Intervall $[i, j]$. Formal

$$\rho(i, j): (\pi_1 \dots \pi_{i-1} \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n) \mapsto (\pi_1 \dots \pi_{i-1} \pi_j \pi_{j-1} \dots \pi_{i+1} \pi_i \pi_{j+1} \dots \pi_n)$$

Geben Sie ein Zertifikat und einen Polynomialzeitverifizierer an. Beschreiben Sie dazu im Detail die Kodierung und die Länge des Zertifikates und die Arbeitsweise des Verifizierers. Die Motivation dieses Problems kommt aus der Bioinformatik: Viele Mutationen auf DNA-Strängen finden ausschließlich durch solche *reversals* statt. Eine kürzeste Folge dieser reversals entspricht wahrscheinlich den Mutationen, die wirklich stattfanden.

Lösungsvorschlag.

Zertifikat und Länge: Das Zertifikat sind die Indizes der reversals. Diese werden binärkodiert und durch entsprechende Trennsymbole separiert. Da höchstens k reversals erlaubt sind, ergibt sich eine Länge von $2k \log n +$ Trennzeichen. *Polynomialzeitverifizierer:* Der Zertifizierer muss testen, ob das Zertifikat die richtige Form hat und ob die Hintereinanderausführung der reversals auf π zu σ führen.

- (1) Hat das Zertifikat die Korrekte Form?
- (2) Führe die Vertauschungen hintereinander auf π aus,
- (3) Vergleiche π und σ .

Laufzeit: In Schritt 1 wird die gesamte Eingabe überprüft, was in Zeit $\mathcal{O}(n)$ erledigt wird. Eine Vertauschung von zwei Elementen realisieren wir wie folgt: Das erste Element wird auf eine extra Spur geschrieben, und danach dieses Element in π überschrieben. Nun kann das zwischengespeicherte Element an die Position des zweiten Elementes geschrieben werden, dies dauert Zeit $\mathcal{O}(\log n)$. Auch müssen wir immer wieder den Zähler welche Elemente nun vertauscht werden müssen aktualisieren ($\mathcal{O}(\log n)$). Für jedes reversal müssen wir schlimmstenfalls $n/2$ viele Vertauschungen durchführen, Also dauert ein reversal $\mathcal{O}(n \log n)$. Dies gibt eine Laufzeit von $\mathcal{O}(k \cdot n \cdot \log n)$ für Schritt 2. Im letzten Schritt müssen wir noch zwei Strings der Länge $\mathcal{O}(n \log n)$ vergleichen. Der Verifizierer läuft folglich in Polynomialzeit.

Aufgabe H22 (10 Punkte)

Beweisen Sie die folgende Aussage: Die Entscheidungsvariante des TRAVELLING SALESMAN PROBLEMS (TSP) ist genau dann in polynomieller Zeit lösbar, wenn dies auch für die Optimierungsvariante gilt. Die Gewichte mögen durch rationale Zahlen in Binärokodierung kodiert sein.

Lösungsvorschlag.

Wir nehmen also an, dass die Entscheidungsvariante des TSP in P ist und konstruieren einen Polynomialzeit-Algorithmus \mathcal{A}_{OPT} , der eine optimale Lösung des TSP berechnet, und dabei den Polynomialzeit-Algorithmus für die Entscheidungsvariante \mathcal{A}_{ENT} als Unterprogramm benutzt.

Als Eingabe erhält der Algorithmus \mathcal{A}_{OPT} einen gewichteten vollständigen Graphen G mit n Knoten. Wir bezeichnen mit d_{max} das maximale Kantengewicht in G . Jede Rundreise in G hat höchstens Kosten von nd_{max} . Folglich können wir eine Kante aus G „löschen“, indem wir ihre Kosten auf $nd_{\text{max}} + 1$ setzen.

Im Folgenden nehmen wir an, dass der Wert b_{opt} der optimalen Lösung durch ein Orakel gegeben ist. Wir können dieses Orakel später durch eine binärer Suche ersetzen und damit b_{opt} in Polynomialzeit finden. Der Algorithmus \mathcal{A}_{OPT} zur Berechnung der optimalen Rundreise geht nun wie folgt vor:

- \mathcal{A}_{OPT} iteriert über alle Kanten e des Graphen G .
- Dabei „löscht“ \mathcal{A}_{OPT} die Kante e und prüft mit Hilfe von \mathcal{A}_{ENT} , ob es eine Rundreise mit Kosten b_{opt} gibt.
 - Falls ja, dann gibt es eine optimale Rundreise, die nicht über e führt. \mathcal{A}_{OPT} „löscht“ die Kante e aus dem Graphen und wählt eine andere Kante bis nur noch n Kanten vorhanden sind.
 - Falls nein, dann führt jede optimale Rundreise über die Kante e und die Kante e wird nicht gelöscht. \mathcal{A}_{OPT} wählt nun eine andere Kante.

Nachdem über alle Kanten iteriert wurde, sind nur noch die Kanten einer optimalen Rundreise im Graphen vorhanden. \mathcal{A}_{OPT} gibt diese als optimale Lösung aus.

Jede Kante wird höchstens einmal aus G entfernt. Dabei wird jeweils der Algorithmus \mathcal{A}_{ENT} einmal aufgerufen. Folglich ist die Laufzeit von \mathcal{A}_{OPT} polynomiell in der Eingabelänge beschränkt.

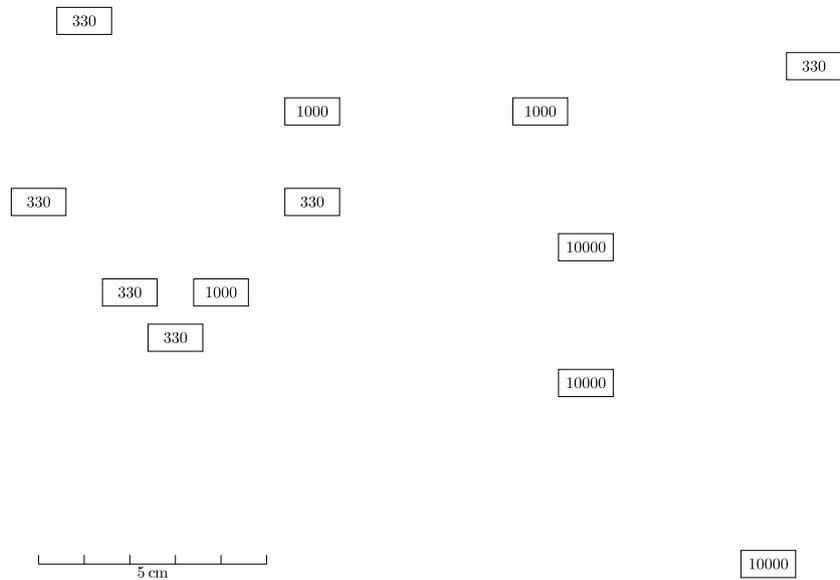
Aufgabe H23 (15 Punkte)

Eine selbstgebaute Bestückungsmaschine kann Widerstände mit verschiedenen Widerstandswerten an die richtigen Plätze in eine Platine stecken. Sie benötigt 2 Sekunden, um auf einen verschiedenen Wert zu wechseln, ansonsten kann sich der Bestückungskopf mit einem Zentimeter pro Sekunde in eine beliebige Richtung bewegen. Das eigentliche Einstecken eines Widerstands benötigt ebenfalls etwas Zeit, aber diese Zeit ist fest und hängt insbesondere nicht vom Widerstandswert ab oder davon, was die Maschine in der Vergangenheit machte.

Die folgende Tabelle zeigt die Positionen (in Zentimetern) und Widerstandswerte (in Ohm) von elf Widerständen, mit welchen eine Platine bestückt werden muß. Sie können

davon ausgehen, daß der Bestückungskopf sich anfangs bereits auf einer der zu bestückenden Positionen befindet und mit dem richtigen Widerstandswert geladen ist. Der erste Widerstand kann also sofort eingesteckt werden. Die Zeichnung neben der Tabelle zeigt, wie die Widerstände am Ende auf der Platine positioniert sein werden (aber nicht im Maßstab 1 : 1).

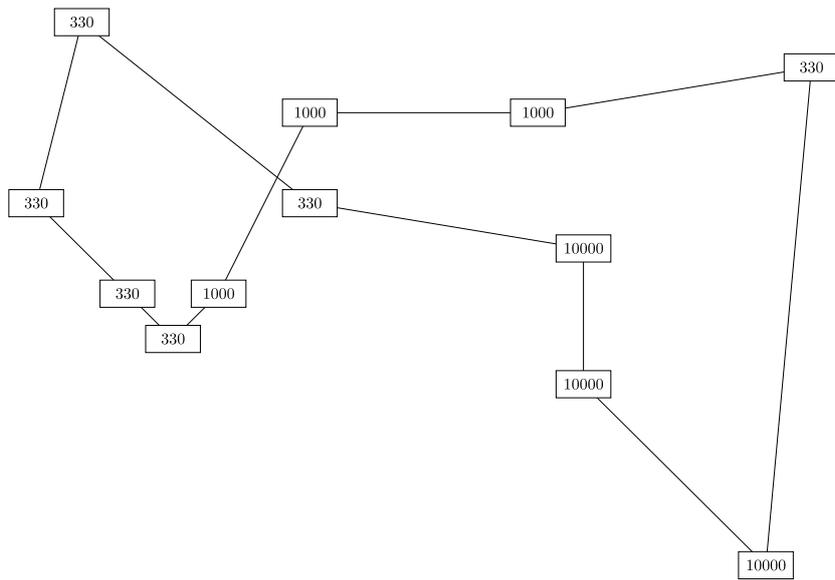
x	y	Wert
2	8	330
3	12	330
4	6	330
5	5	330
6	6	1000
8	10	1000
8	8	330
13	10	1000
14	4	10000
14	7	10000
18	0	10000
19	11	330



- Finden Sie eine optimale Reihenfolge, in welcher alle Widerstände bestückt werden können und der Bestückungskopf anschließend wieder in die Ausgangslage zurückkehrt und wieder mit dem gleichem Widerstandswert wie am Anfang geladen wird. (So kann eine Serie von Platinen nacheinander auf die gleiche Weise bestückt werden.)
- Wie können Sie garantieren, daß Ihre Lösung optimal ist?
- Schätzen Sie, mit bis zu wievielen Widerständen man Ihre Lösungsmethode noch verwenden könnte, bevor die Laufzeit unpraktikabel groß würde.
- Haben Sie eine Idee, was Sie machen würden, wenn es um hunderte von Widerständen geht?

Hinweis: Natürlich dürfen Sie ein Programm schreiben, welches bei der Lösung dieser Aufgabe hilft.

Lösungsvorschlag.



Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T21

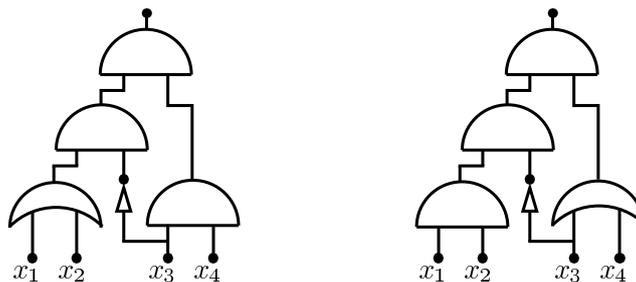
Sie wollen zur Festzeit ihr Haus mit bunten Lampen dekorieren. Leider haben Sie ihren guten Freund, einen Elektrotechnikstudenten, um Hilfe gebeten und nun ist die Beleuchtung derart kompliziert, daß Sie Schwierigkeiten haben, sie überhaupt anzuschalten.

Ein Boolesches Schaltnetz besteht aus einem gerichteten, azyklischen Graphen mit Eingangsknoten x_1, \dots, x_n sowie einem Ausgangsknoten x_{out} . Die Eingangsknoten haben dabei nur eine ausgehende Kanten und der Ausgangsknoten nur eine eingehende Kante. Alle weiteren Knoten sind markiert als ODER-, UND- oder NICHT-Gatter, außerdem sind Verzweigungen möglich.



Für das Entscheidungsproblem BOOLEAN CIRCUIT ist ein solches Schaltnetz gegeben und die zugehörige Frage ist, ob es eine erfüllende Belegung gibt.

Die folgenden Schaltungen hat ihr Freund in die Beleuchtung verbaut, mit der Behauptung, daß die jeweilige Lichterkette angeht, wenn man die Schalter x_1, \dots, x_4 in die richtige Stellung bringt. Ist dies tatsächlich für beide Schaltungen möglich? Wenn ja, geben Sie eine Schalterstellung an, die ihr Haus festlich beleuchtet.



Lösungsvorschlag

Die erste Schaltung ist unerfüllbar. Die zweite Schaltung ist erfüllbar durch $x = 1101$.

Aufgabe T22

Reduzieren Sie das oben beschriebene Problem BOOLEAN CIRCUIT auf das SAT-Problem aus der Vorlesung. Formal: Zeigen Sie $\text{BOOLEAN CIRCUIT} \leq_p \text{SAT}$. Denken Sie dabei an Zimt, Tannengeruch und Kerzenschein.

Lösungsvorschlag.

Wir werden jedes AND-Gatter und auch jedes OR-Gatter als eine Aussagenlogische Formel φ in CNF darstellen, daraus kann dann eine Formel Φ für die gesamte Schaltung erstellt werden. Zuerst betrachten wir ein AND-Gatter. Seien $x_1 \dots x_n$ die Eingänge des Gatters und S der Ausgang. S ist genau dann erfüllt, wenn alle x_i erfüllt sind, sprich ϕ hat das Aussehen $\varphi = S \leftrightarrow x_1 \wedge \dots \wedge x_n$. Lösen wir den Äquivalenzpfeil auf, erhalten wir $\varphi = (S \leftarrow x_1 \wedge \dots \wedge x_n) \wedge (S \rightarrow x_1 \wedge \dots \wedge x_n)$. Aus der Aussagenlogik wissen wir daß $A \rightarrow B := \neg A \vee B$ gilt. Wenden wir dies auf φ an, erhalten wir $\varphi = (\neg S \vee (x_1 \wedge \dots \wedge x_n)) \wedge (S \vee \neg(x_1 \wedge \dots \wedge x_n))$. Wenden wir nun noch die DeMorgan'schen Gesetze an, erhalten wir $\varphi = (\neg S \vee x_1) \wedge \dots \wedge (\neg S \vee x_n) \wedge (\neg x_1 \vee \dots \vee \neg x_n \vee S)$. Analog kann für ein OR-Gatter verfahren werden. Die einzelnen Formeln φ werden nun wie in der Schaltung "zusammengesteckt" zur Formel Φ .

Aufgabe T23

Das Problem PLANAR BOOLEAN CIRCUIT ist definiert wie das BOOLEAN CIRCUIT-Problem, jedoch dürfen sich nun keine zwei Drähte kreuzen. Zeigen Sie:

$$\text{BOOLEAN CIRCUIT} \leq_p \text{PLANAR BOOLEAN CIRCUIT}$$

Denken Sie dabei darüber nach, ob Sie nicht doch noch Geschenke besorgen müssen: wenn Sie dieses Blatt lesen, ist die Zeit schon recht knapp!

Aufgabe H24 (10 Punkte)

Sie müssen die Beleuchtung ihres Hauses reparieren, aber aufgrund einer Knappheit an seltenen Erden sind NICHT-Gatter dieses Jahr ausgesprochen teuer. Sie planen daher, NICHT-Gatter durch schlaue Modifizierung der Schaltung zu ersetzen. Als Hilfsmittel haben sie zudem Schalter, die bereits ein NICHT-Gatter eingebaut haben.

Das MONOTONE BOOLEAN CIRCUIT-Problem ist wie BOOLEAN CIRCUIT definiert, aber es darf keine Negationsgatter außer direkt an den Eingängen geben.

Beweisen Sie, daß es eine Möglichkeit gibt, ihre Weihnachtsbeleuchtung ohne NICHT-Gatter zu realisieren, indem Sie zeigen, daß

$$\text{BOOLEAN CIRCUIT} \leq_p \text{MONOTONE BOOLEAN CIRCUIT}$$

gilt.

Aufgabe H25 (10 Punkte)

Ihr E-Technikerfreund behauptet, er könne Schaltungen mit gleicher Funktion auf einen Blick erkennen. Das Problem BOOLEAN CIRCUIT EQUIVALENCE ist formal wie folgt definiert: als Eingabe erhalten wir zwei Boolesche Schaltkreise und müssen entscheiden, ob die beiden Schaltkreise für alle Eingaben die gleiche Ausgabe liefern.

Zeigen Sie, daß ihr Freund entweder ein Wunderkind oder ein Angeber ist, indem Sie beweisen, daß die folgenden zwei Aussagen gelten:

$$\overline{\text{BOOLEAN CIRCUIT EQUIVALENCE}} \leq_p \text{BOOLEAN CIRCUIT}$$
$$\text{BOOLEAN CIRCUIT} \leq_p \overline{\text{BOOLEAN CIRCUIT EQUIVALENCE}}$$

Lösungsvorschlag.

TODO: anpassen. $\overline{\text{BOOLEAN CIRCUIT EQUIVALENCE}} \leq_p \text{BOOLEAN CIRCUIT}$

Zwei Schaltnetze A und B sind equivalent genau dann wenn es keine Belegung der Variablen gibt, sodass sie sich im Akzeptanzverhalten unterscheiden. Hierzu kann einfach eine neue Schaltung aus den beiden gegebenen Schaltnetzen konstruiert werden. $A \oplus B$ (A XOR B) ist genau dann wahr, wenn sich A und B in ihrem Akzeptanzverhalten unterscheiden. Damit Ja-Instanzen von Boolean Circuit Equivalence auch auf Ja-Instanzen von Boolean Circuit abgebildet werden, muss das Ergebnis noch negiert werden. Das heißt A und B sind equivalent genau dann wenn $\overline{A \oplus B}$ erfüllbar ist.

$\text{BOOLEAN CIRCUIT} \leq_p \overline{\text{BOOLEAN CIRCUIT EQUIVALENCE}}$

Konstruiere einen unerfüllbaren Schaltkreis U , für die gleiche Anzahl an Variablen wie der in BOOLEAN CIRCUIT gegebene Schaltkreis A . Nun ist (A, E) äquivalent genau dann wenn A nicht erfüllbar ist.

Aufgabe H26 (10 Punkte)

Der Weihnachtsmann hat m Geschenke, die er an n liebe Kinder verteilen möchte. Jedes Kind kann dabei mehrere Geschenke bekommen, jedoch mag jedes Kind die unterschiedlichen Geschenke unterschiedlich gerne. Ein Kind i möge also ein Geschenk j genau $p_{i,j} \in \mathbb{N}$ gerne (diese hypothetischen Kinder haben äußerst präzise Wunschzettel).

Die Freude eines Kindes i sei dabei als $\mathcal{F}(i) = \sum_{j \in G_i} p_{i,j}$ definiert, wobei $G_i \subseteq \{1, \dots, m\}$ die Geschenke bezeichnet, die das Kind i erhält. Natürlich möchte der Weihnachtsmann allen Kinder eine möglichst glückliche Weihnachtszeit beschern, daher versucht er die Geschenke so in Mengen G_1, \dots, G_n einzuteilen, daß die Freude des traurigsten Kindes maximal ist. Formal will er also die Funktion $c(\mathcal{G}) = \min_{1 \leq i \leq n} \mathcal{F}(i)$ maximieren, wobei \mathcal{G} eine beliebige Partition der m Geschenke bezeichne.

Bei der Entscheidungsvariante des SANTA CLAUS PROBLEMS soll entschieden werden, ob eine gegebene Mindestweihnachtsfreude k erreicht werden kann, also, ob es eine Lösung \mathcal{G} gibt, für die $c(\mathcal{G}) \geq k$ gilt.

Zeigen Sie, daß das SANTA CLAUS PROBLEM NP-vollständig ist.

Hinweis: Für die Reduktion empfehlen wir das NP-vollständige Problem PARTITION, das wie folgt definiert ist:

Gegeben ist eine endliche Menge natürlicher Zahlen $A = \{a_1, \dots, a_n\} \subset \mathbb{N}$. Kann A so in zwei disjunkte Teilmengen zerlegt werden, daß die Summe der Elemente in beiden Teilmengen gleich groß ist?

Lösungsvorschlag.

Santa Claus Problem \in NP kann leicht gezeigt werden indem als Zertifikat eine Verteilung der Geschenke auf die Kinder angegeben wird. Werden die Mengen die jedes Kind bekommt im Zertifikat durch ein Trennsymbol getrennt hintereinander aufgeschrieben, dann wird $O(n + m \cdot \log m)$ Platz benötigt.

(z.B.: die Form $z =$ Geschenke für Kind 1#Geschenke für Kind 1#... braucht n Rauten und für jedes der m Geschenke eine $(\log m)$ -stellige Binärzahl)

Man kann leicht eine Instanz von Partition in eine Instanz des Santa Claus Problem umwandeln indem man $n = 2$, $p_{1,j} = p_{2,j} = a_j$ und $k = \frac{1}{2} \sum_j a_j$ setzt. Die so konstruierte

Santa Claus Instanz kann genau dann gelöst werden, wenn die Partition Instanz lösbar ist.

