Lösungen zu Übungen
Berechenbarkeit und Komplexität WS00/01 (Thomas)

erstellt von
Björn Eickvonder

1. Juni 2001

Teil I

Lösungen zur Anwesenheitsübung zu Berechenbarkeit und Komplexität WS00/01

Aufgabe 1

(a)

Beweis per Induktion über l:

Induktionsanfang: $l = 0$
Das einzige Wort mit der Länge $l = 0$ ist ε, also $|\,\varepsilon\,| = 0$, nun gilt aber noch:

$$m^0 = 1$$

Induktionsvoraussetzung:

$$\exists\, m^l \text{ Worte der Länge } l$$

Induktionsschluß: $l \to l + 1$
Da es über dem Alphabet Σ_m genau m verschiedene Zeichen gibt folgt, dass die m-fache Anzahl der Wörter entsteht, wenn die Länge l um 1 erhöht wird, also:

$$m^l \cdot m = m^{l+1} \quad \square$$

(b)

Beweis per Induktion über l:

Induktionsanfang: $l = 0$
Das einzige Wort mit der Länge $l = 0$ ist ε, also $|\,\varepsilon\,| = 0$, nun gilt aber noch:

$$\frac{m^1 - 1}{m - 1} = 1$$

Induktionsvoraussetzung:

$$\exists\, \frac{m^{l+1} - 1}{m - l} \text{ Worte der Länge } \leq l$$

Induktionsschluß: $l \to l + 1$
Da es über dem Alphabet Σ_m für die Länge $l + 1$ nach (a) genau m^{l+1} Wörter gibt folgt:

$$\frac{m^{l+1} - 1}{m - 1} + m^{l+1} = \frac{m^{l+1} - 1 + m^{l+2} - m^{l+1}}{m - 1} = \frac{m^{l+2} - 1}{m - 1} \quad \square$$

Aufgabe 2

Wähle $2^r \geq m$ und sei $bin(i)$ Binärdarstellung der Zahl i und $\underline{i} = i$-ter Buchstabe des Alphabets Σ_m, dann sind folgende Funktionen eine Kodierung im Sinne der Aufgabe:

$$\begin{aligned} \mathrm{f}_1(\underline{\mathrm{i}}) &= bin(i) \text{ der Länge } r \text{ (links aufgefüllt durch 0-en)} \\ \mathrm{f}_2(\underline{\mathrm{i}}) &= \underbrace{0\ldots0}_{i}1 \end{aligned}$$

Aufgabe 3

Sei $\Sigma = \{|, 0, \ldots, 9, a, b\}$, dann läßt sich der Transitionsgraph der Aufgabe wie folgt kodieren:

$$\underbrace{1}_{\text{Startknoten}} \quad || \quad \underbrace{1a1|1b2|2b1|2a2}_{\text{Kanten}} \quad || \quad \underbrace{2}_{\text{Endknoten}}$$

ACHTUNG: Gibt es isolierte Knoten, werden diese durch die obige Kodierung nicht erfaßt!

Teil II

Lösungen zu Übungen zu Berechenbarkeit und Komplexität WS00/01

Serie 1

Aufgabe 1

(a)

$$\begin{aligned}\gamma_3(\underline{1}\,\underline{2}) &= 2\cdot 3^0 + 1\cdot 3^1 = 5\\ \gamma_3(\underline{3}\,\underline{1}\,\underline{3}\,\underline{2}) &= 2\cdot 3^0 + 3\cdot 3^1 + 1\cdot 3^2 + 3\cdot 3^3 = 101\end{aligned}$$

(b)

$$\begin{array}{l} \underline{\delta_3(12)} \\ 12 = 3\cdot 3 + 3 \\ 3 = 0\cdot 3 + 3 \\ \Rightarrow \delta_3(12) = \underline{3}\,\underline{3} \end{array}$$

$$\begin{array}{l} \underline{\delta_3(79)} \\ 79 = 26\cdot 3 + 1 \\ 26 = 8\cdot 3 + 2 \\ 8 = 2\cdot 3 + 2 \\ 2 = 0\cdot 3 + 2 \\ \Rightarrow \delta_3(79) = \underline{2}\,\underline{2}\,\underline{2}\,\underline{1} \end{array}$$

Aufgabe 2

(a)

$$a_1\dots a_l <_{lex} b_1\dots b_k \quad\Leftrightarrow\quad \begin{array}{l}\Big(\exists\, z \le min\{l,k\} \;\big|\; \gamma(a_i)\le\gamma(b_i) \quad \forall\, i\le z \;\wedge\; \big(\exists\, i\le z \mid \gamma(a_i)<\gamma(b_i)\big)\Big)\\ \vee\Big(\gamma(a_i)=\gamma(b_i) \quad \forall\, i\le l \;\wedge\; k>l\Big)\end{array}$$

Ein Wort $a_1\dots a_l$ ist lexikographisch kleiner als ein Wort $b_1\dots b_k$, wenn es einen ersten Buchstaben in a von links gesehen gibt, der in der kanonischen Reihenfolge des Alphabets kleiner ist als der entsprechende Buchstabe in b oder wenn alle Buchstaben gleich sind und Wort b eine Verlängerung von Wort a darstellt.

(b)

$$\Sigma_2 = \{\underline{1};\underline{2}\} \quad\Rightarrow\quad \underline{2} \;>_{lex}\; \underline{1}\,\underline{2} \;>_{lex}\; \underline{1}\,\underline{1}\,\underline{2} \;>_{lex}\; \underline{1}\,\underline{1}\,\underline{1}\,\underline{2} \;>_{lex}\; \dots$$

Serie 2

Aufgabe 3

Sei A der Algorithmus, der W aufzählt. Gesucht ist ein Algorithmus B, der W wiederholungsfrei aufzählt. Sei L eine leere Liste, dann arbeitet B wie folgt:

1. Sei w das nächste (bzw. erste) Ausgabewort von A. Dann prüfe, ob w in der Liste enthalten ist.
   ja $\rightarrow$ fahre fort bei 1.
   nein $\rightarrow$ fahre fort bei 2.
2. gib w aus (zähle w auf), trage w in die Liste ein und fahre bei 1 fort.

Aufgabe 4

$\Rightarrow$:

Sei A der Algorithmus, der W entscheidet. Bearbeite wie folgt:

1. Setze $w = \varepsilon$.

2. Sei $t =$ Ausgabe von A bei Eingabe von w.
   $t =$ ja $\to$ fahre fort bei 3.
   $t =$ nein $\to$ fahre fort bei 4.

3. Gebe w aus und fahre fort bei 4.

4. Setze $w =$ nächstes Wort in kanonischer Reihenfolge aus Σ^* und fahre fort bei 2.

$\Leftarrow$:

Sei A der Algorithmus, der W monoton aufzählt und $u \in \Sigma^*$ eine beliebige Eingabe für die entschieden werden soll, ob sie in W enthalten ist. Bearbeite wie folgt:

1. Setze $w =$ erstes Ausgabewort von A.

2. Ist $w = u$?
   ja $\to$ terminiere mit ja.
   nein $\to$ fahre fort bei 3.

3. Ist $w < u$ in kanonischer Reihenfolge ?
   ja $\to$ Setze $w =$ nächstes Ausgabewort von A und fahre fort bei 2.
   nein $\to$ terminiere mit nein.

Aufgabe 5

Idee:

Programm P terminiert nicht, wenn es ein zweites Mal mit der gleichen Variablenbelegung eine Programmzeile erreicht. Führe während der Berechnung von P in einer Liste Buch über die Speicherzustände, mit denen die einzelnen Zeilen aufgerufen werden. P habe p Programmzeilen und n Variablen $X1, \ldots, Xn$. Dann werden die Zustände beschrieben durch $(q, v_1, \ldots, v_n)$ mit $q \in \{1, \ldots, p\}$ und $v_i =$ Wert der Variablen Xi bei Aufruf von Programmzeile q. Es gibt nur endlich viele solcher Tupel (P hat endlich viele Programmzeilen p, Variablen n, Variablenwerte). Der Algorithmus, der das Problem entscheidet, arbeitet mit Hilfe einer zunächst leeren Liste wie folgt:

1. Bei jedem Berechnungsschritt von P wird überprüft, ob der Zustand schon in der Liste eingetragen ist.
   ja $\to$ terminiere mit nein.
   nein $\to$ trage Zustand in die Liste ein und fahre fort bei 1.

Terminiert P irgendwann so terminiere mit ja.

Serie 3

Aufgabe 6

Folgende Turingmaschine $M = (\{q_0, q_1, q_s\}, \{0, 1\}, \{0, 1, \sqcup\}, q_0, q_s, \delta)$ berechnet die Funktion der Aufgabe.

δ :

q_0	0	0	R	q_0	ans Ende des Wortes ohne invertieren
q_0	1	1	R	q_0	
q_0	$\sqcup$	$\sqcup$	L	q_1	
q_1	0	1	L	q_1	zum Anfang des Wortes und dabei invertieren
q_1	1	0	L	q_1	
q_1	$\sqcup$	$\sqcup$	R	q_s	

Konfigurationsfolge von M auf Eingabe 001:

$$\begin{aligned} q_0001 &\underset{M}{\vdash} 0q_001 \underset{M}{\vdash} 00q_01 \underset{M}{\vdash} 001q_0\sqcup \underset{M}{\vdash} 00q_11 \\ &\underset{M}{\vdash} 0q_100 \underset{M}{\vdash} q_1010 \underset{M}{\vdash} q_1 \sqcup 110 \underset{M}{\vdash} q_s110 \end{aligned}$$

Aufgabe 7

$$\delta'(q_i, a) = \begin{cases} \delta(q_i, a) & \text{falls } \delta(q_i, a) = (a, R|L, q_j) \\ (a, R, q_j') & \text{falls } \delta(q_i, a) = (a, N, q_j) \end{cases}$$

$$\begin{aligned} &\delta'(q_j',) = (x, L, q_j) \quad \forall x \in \Gamma \\ &Q' = Q \cup \{q' \mid q \in Q\} \qquad \Gamma' = \Gamma \qquad q_0' = q_0 \qquad q_s' = q_s \end{aligned}$$

Aufgabe 8

$\Rightarrow$:

Sei A der Algorithmus, der W aufzählt. Gesucht ist ein Algorithmus B, der W semi-entscheidet. Sei $w \in \Sigma^*$ eine beliebige Eingabe, so arbeitet B wie folgt:

1. Setze u = nächstes (erstes) Ausgabewort von A
2. Ist $u = w$?
   ja $\rightarrow$ terminiere mit ja.
   nein $\rightarrow$ fahre fort bei 1

$\Leftarrow$:

Sei A der Algorithmus, der W semi-entscheidet. Gesucht ist ein Algorithmus B, der W aufzählt. B arbeitet wie folgt:

- ☞ Seien $w_i \in \Sigma^*$ Worte in kanonischer Reihenfolge, so dass $w_1 = \varepsilon$. Lasse nun A auf w_1 zwei Schritte laufen, dann starte A erneut mit der Eingabe w_2 und lasse ihn einen Schritt laufen, dann führe einen weiteren Schritt für w_1 aus, dann einen weiteren Schritt für w_2, dann starte w_3 und lasse ihn einen Schritt laufen, dann lasse w_1 wieder einen Schritt laufen, ...
- ☞ Terminiert A für ein w_i, so gebe w_i aus und fahre fort.

Serie 4

Aufgabe 9

```
X3 := X3 + 1;
if X1 > 0 then
  X3 := X3
else
  while X3 > 0 do X3 := X3 end /*while*/
end; /*if*/
if X2 > 0 then
  X3 := X3
else
  while X3 > 0 do X3 := X3 end /*while*/
end; /*if*/
while X3 > 0 do
  X3 := X1  mod  X2;
  X1 := X2;
  X2 := X3;
end /*while*/
```

Aufgabe 10

(a)

```
loop X1 begin
  X2 := X2 + 1;
  X3 := X3 + X2
end; /*loop*/
X1 := X3
```

(b)

```
loop X1 begin
  X3 := X2 * X2;
  if X3 > X1 then
    X4 := X4
  else
    X4 := X2;
    X2 := X2 + 1
  end; /*if*/
end; /*loop*/
X1 := X4
```

Aufgabe 11

Die Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) = n^2$ ist LOOP-berechenbar durch folgendes Programm P:

`P = X1 := X1 * X1`

Ein LOOP-freies additives Programm (kurz: LFA-Programm) P hat nur Anweisungen der Form:

☞ Wertzuweisungen:

1. `Xi := 0`
2. `Xi := Xj`
3. `Xi := Xj + 1`
4. `Xi := Xj - 1`
5. `Xi := Xj + Xk`
6. `Xi := Xj - Xk`

☞ Komposition:
Wenn P_1 und P_2 LFA-Programm $\Rightarrow$ P_1;P_2 LFA-Programm

☞ if-then-else Konstrukt

Behauptung (Eigenschaft):

Ein LFA-Programm P mit l Anweisungen, das eine einstellige Funktion $f : \mathbb{N} \to \mathbb{N}$ berechnet, kann für eine Eingabe $n \in \mathbb{N}$ höchstens einen Wert $\leq n \cdot 2^l$ (für $n = 0$ einen Wert $\leq 2^{l-1}$) in einer seiner Variablen annehmen.

Bemerkung:

Habe P_1 l_1 Anweisungen und P_2 l_2 Anweisungen, dann gibt es zu jeder Anweisung "if B then P_1 else P_2 end" ein $l' = max\{l_1, l_2\}$. Somit besitzt ein LFA-Programm P eine maximale Anzahl l von Wertzuweisungen, die nicht von der Eingabe abhängen.

Beweis: (per Induktion über l)

Induktionsanfang: $l = 1$

☞ 1. Fall $X1 = 0$
maximaler Wert wird erreicht durch $P = Xi := Xi + 1$ für i beliebig, d.h. $Xi = 1 \leq 2^0$

☞ 2. Fall $X1 > 0$
maximal erzielbarer Wert kann nur unter Verwendung von $X1$ erreicht werden, d.h. durch
$P : Xi := X1 + X1 \Rightarrow Xi = X1 + X1 = 2 \cdot X1 = n \cdot 2^1$

Induktionsvoraussetzung:

Für ein LFA-Programm P mit l Anweisungen gilt:
Der maximal erzielbare Wert einer Variablen ist $\leq n \cdot 2^l$ bzw. $\leq 2^{l-1}$ (für $n = 0$).

Induktionsschluß: $l \rightarrow l + 1$
Mit einer Anweisung P_2 gilt dann, daß $P = P_1; P_2$ $(l+1)$-Wertzuweisungen hat. Der maximal mit P erreichbare Wert einer Variablen (ist nur mit $(l + 1)$-Anweisungen der Form $Xj := Xi + Xi$ erreichbar) ist:

$$\leq 2 \cdot Xi \overset{IV}{=} 2n \cdot 2^l = n \cdot 2^{l+1} \text{ bzw. } \leq 2 \cdot Xi \overset{IV}{=} 2 \cdot 2^{l-1} = 2^l \text{ (für } n = 0).$$

Somit kann $X1$ auch höchstens den Wert $n \cdot 2^l$ bzw. 2^{l-1} (für $n = 0$) annehmen. Also:

$$f_p^{(1)}(n) \leq n \cdot 2^l \text{ bzw. } f_p^{(1)}(n) \leq 2^{l-1} \text{ (für } n = 0)$$

$\Rightarrow f(n) = n^2$ kann nicht durch ein LFA-Programm berechnet werden, da n^2 schneller wächst als $n \cdot 2^l$, da 2^l eine feste Konstante ist. □

Serie 5

Aufgabe 12

(a)

```
X2 := X1 - 1;
if X2 > 0 then
  X3 := X3 + 1; /*X3 speichert f(i-2)*/
  X4 := X4 + 1; /*X4 speichert f(i-1)*/
  loop X2 begin
    X5 := X3 + X4;
    X3 := X4;
    X4 := X5
  end; /*loop*/
  X1 := X5
else
  X1 := 0;
  X1 := X1 +1
end
```

(b)

```
X2 := X2 + 1;
X3 := X3 + 1;
X3 := X3 + 1;
loop X1 begin
  X2 := X2 * X3
end;
X1 := 0;
X1 := X1 + 1;
loop X2 begin
  X1 := X1 * X3
end
```

Die erste Schleife berechnet $X2 = 2^n$, die zweite dann $X1 = 2^{(2^n)}$.

Aufgabe 13

Sei P ein WHILE-Programm mit if-then-else-Konstrukt und sei $maxind(P)$ = maximal vorkommender Variablenindize im Programm P und $m = max\{\text{Anzahl der Eingabevariablen von } P, maxind(P)\} + 1$. Dann ersetze jedes if-then-else Konstrukt

```
if Xi > 0 then P1 else P2 end
```

in P durch

```
Xm := Xi;
while Xm > 0 do
  Xm := 0; P1
end;
Xm := 1 - Xi; (eigtl. X(m+1) := X(m+1) + 1; Xm := X(m+1) - Xi;)
while Xm > 0 do
  Xm := 0; P2
end
```

Aufgabe 14

Nach Voraussetzung gibt es ein LOOP-Programm P_f, das f berechnet. Sei $m = maxind(P_f)$ definiert wie in Aufgabe 13, dann berechnet F folgendes Programm P:

```
P = loop X2 begin
      X2 := 0; X3 := 0; X4 := 0; ...; Xm := 0; Pf
    end
```

Beachte: Die Veränderung der Variablen $X2$ innerhalb der Schleife hat keinen Einfluß auf die Anzahl der Iterationen.

Serie 6

Aufgabe 15

Folgendes WHILE$^+$-Programm P berechnet f

```
P = X2 := X2 + 1;
    X2 := X2 + 1;
    X3 := X2 - X1; /*X3>0, falls X1=0 oder X1=1*/
    if X3 > 0 then
      while X2 > 0 do X2 := X2 end /*0 und 1 keine Primzahl -> Endlosschleife*/
    else
      while X2 < X1 do
        X3 := X1 mod X2;
        if X3 > 0 then
          X3 := X3
        else
          X2 := X2 - 1
        end;
        X2 := X2 + 1
      end;
      X1 := 0;
      X1 := X1 + 1
    end
```

Die Haupt-WHILE-Schleife überprüft zunächst ob der Eingabewert $X1$ durch $X2$ (zunächst vom Wert 2) teilbar ist. Ist dies nicht der Fall wird der Zähler $X2$ um 1 erhöht und es wird erneut auf Teilbarkeit überprüft, solange bis $X2 = X1$ ist, dann terminiert das Programm mit 1. Ergibt jedoch eine Überprüfung für ein $X2$, dass $X1$ durch $X2$ teilbar ist, wird der Zähler $X2$ zunächst um 1 vermindert, um dann wieder um 1 erhöht zu werden, wodurch beim nächsten Schleifendurchlauf wieder mit dem selben $X2$-Wert gearbeitet wird. $\rightarrow$ Endlosschleife $\rightarrow$ das Programm terminiert nicht ($\rightarrow f(n) = \perp$).

Aufgabe 16

(a)

Angenommen es gilt $D = R_a$ für ein $a \in A$. Dann ist

$$a \in D \underset{R_a = D}{\iff} a \in R_a \underset{\text{Def.}R_a}{\iff} (a,a) \in R \underset{\text{Def.}D}{\iff} a \notin D \qquad \textbf{Widerspruch !}$$

(b)

zz.: Für jede entscheidbare Wortmenge L über Σ_{bool} gibt es ein $u \in \Sigma_{bool}$ mit $L = R_u$

Aus der Definition für R_a in (a) ergibt sich:

$$R_u = \begin{cases} \{v \in \Sigma_{bool}^*\} \mid M_u \text{ stoppt nicht angesetzt auf } v & \text{falls } u \text{ Kodewort der Turingmschine } M_u \\ \Sigma_{bool}^* & \text{falls } u \text{ nicht Kodewort irgendeiner Turingmaschine} \end{cases}$$

Sei $L \subseteq \Sigma_{bool}^*$ entscheidbar. Dann ist $\overline{L} := \Sigma_{bool}^* \backslash L$ entscheidbar und somit auch semi-entscheidbar. Sei u das Kodewort einer Turingmaschine M_u, die $\overline{L}$ semi-entscheidet, dann stoppt M_u genau auf den Wörtern aus L nicht. Also gilt $L = R_u$

(ACHTUNG: EIGENE LÖSUNG! - evtl. unvollständig)

Sei D definiert wie in (a). Dann ist D die Menge aller binären Wörter b, die Kodewort einer Turingmaschine sind, die angesetzt auf sich selbst stoppen. D ist nicht Turing-entscheidbar, da es eine Turingmaschine M geben müßte, die für ein Kodewort einer weiteren Turingmaschine entscheidet, ob diese auf sich selbst angesetzt stoppt. Diese gibt es aber nicht, da sie auch über sich selbst Auskunft geben müßte.

Aufgabe 17

keine Lösung vorhanden.

Serie 7

Aufgabe 18

(a) (P1)

(P1) ist entscheidbar. Folgender Algorithmus entscheidet das Problem:
Lasse M angesetzt auf w (maximal) n Schritte laufen. Gelangt M innerhalb der n Schritte in den Zustand q_s, so gebe 1 aus, sonst gebe 0 aus.

(b) (P2)

(P2) ist nicht entscheidbar.
Angenommen (P2) wäre entscheidbar, dann würde folgender Algorithmus mit den Eingaben M, w, n das Wortproblem <u>WP</u> lösen:

1. Lasse Algorithmus, der (P1) entscheidet mit Eingaben M, w, n laufen.
   Bei Ausgabe 1 $\rightarrow$ terminiere mit 1.
   Bei Ausgabe 0 $\rightarrow$ fahre fort mit 2.

2. Lasse Algorithmus, der (P2) entscheidet (existiert nach Annahme) mit M, w, n laufen.
   Bei Ausgabe 1 $\rightarrow$ terminiere mit 1.
   Bei Ausgabe 0 $\rightarrow$ terminiere mit 0.

Widerspruch zur Unentscheidbarkeit des Wortproblems <u>WP</u> !

Aufgabe 19

1. Halteproblem $\underline{H} = (I_H, H)$ mit
   $I_H =$ Menge der Turingmaschinen M über $\{0,1\}$
   $H = \{M \in I_H \mid M : \varepsilon \to \mathit{stop}\}$
2. $\underline{H}_{111} = (I_{H_{111}}, H_{111})$ mit
   $I_{H_{111}} =$ Menge der Turingmaschinen M über $\{0,1\}$
   $H_{111} = \{M \in I_{H_{111}} \mid M : 111 \to \mathit{stop}\}$

(a) zz.: $\underline{H}_{111} \leq \underline{H}$

Gesucht ist eine berechenbare Transformation:

$$\begin{aligned} f : I_{H_{111}} \to I_H \text{ mit } x \in H_{111} \Leftrightarrow f(x) \in H \quad \forall x \in I_{H_{111}} \\ \text{d.h.: } f : M \to M' \text{ mit } M : 111 \to \mathit{stop} \Leftrightarrow M' : \varepsilon \to \mathit{stop} \end{aligned}$$

Konstruiere M' wie folgt:
M' schreibt zunächst 111 auf das Band, geht zurück an den Anfang des Wortes 111 und arbeitet dann wie M.
Nach Konstruktion von M' ist klar, dass $M : 111 \to \mathit{stop} \Leftrightarrow M' : \varepsilon \to \mathit{stop}$

(b) zz.: $\underline{H} \leq \underline{H}_{111}$

Analoger Beweis wie zu (a). Konstruktion von M vermutlich: Löschen von 111 auf dem Band, dann weiter wie M'.

Aufgabe 20

Gegeben ist eine eingabebeschränkte Turingmaschine $M = (Q, \{0,1\}, \Gamma, q_0, q_s, \delta)$, $w \in \{0,1\}^*$, Anfangskonfiguration $\text{¢}\, q_0 w \text{¢}$ und Endmarker nur folgende Turingzeilen erlaubt: $q\text{¢}\,\text{¢}\,Rq'$, $q\$\$Lq'$
Es gibt nur endlich viele Konfigurationen:

- ☞ Anzahl der möglichen Bandinschriften: $|\,\Gamma\,|^{|\,w\,|}$
- ☞ Anzahl der möglichen Zustände: $|\,Q\,|$
- ☞ Anzahl der möglichen Positionen des Kopfes: $|\,w\,| + 2$

$\Rightarrow$ Anzahl der möglichen Konfigurationen: $n := |\,\Gamma\,|^{|\,w\,|} \cdot |\,Q\,| \cdot (|\,w\,| + 2)$

Das Problem "Stoppt M von Anfangskonfiguration $\text{¢}\, q_0 w \$$" ist entscheidbar!

Folgender Algorithmus entscheidet das Problem:

Lasse M laufen und merke jede angenommene Konfiguration in einer Liste. Wird eine Konfiguration zum zweiten Mal angenommen (dies geschieht nach spätestens $n+1$ Schritten), so gebe 0 aus. Wird der Stopzustand q_s angenommen, so gebe 1 aus.

Serie 8

Aufgabe 21

(a)

zz. $\underline{P_1} \leq \underline{P_1} \oplus \underline{P_2}$

gesucht ist eine berechenbare Transformation $f : \mathbb{N} \to \mathbb{N}$, so dass gilt:

$$n \in P_1 \Leftrightarrow f(n) \in P_1 \oplus P_2 \quad \forall n \in \mathbb{N}$$

$f(n) := 2n$ erfüllt dies, denn

$$\begin{aligned} \text{Sei } n \in P_1 &\Leftrightarrow f(n) = 2n \in \{2n \mid n \in P_1\} \\ &\Leftrightarrow f(n) \in \{2n \mid n \in P_1\} \cup \{2n+1 \mid n \in P_2\} \\ &\Leftrightarrow f(n) \in P_1 \oplus P_2 \end{aligned}$$

zz. $\underline{P_2} \leq \underline{P_1} \oplus \underline{P_2}$

gesucht ist eine berechenbare Transformation $f : \mathbb{N} \to \mathbb{N}$, so dass gilt:

$$n \in P_2 \Leftrightarrow f(n) \in P_1 \oplus P_2 \quad \forall n \in \mathbb{N}$$

$f(n) := 2n + 1$ erfüllt dies, denn

$$\begin{aligned} \text{Sei } n \in P_2 &\Leftrightarrow f(n) = 2n+1 \in \{2n+1 \mid n \in P_2\} \\ &\Leftrightarrow f(n) \in \{2n+1 \mid n \in P_2\} \cup \{2n \mid n \in P_1\} \\ &\Leftrightarrow f(n) \in P_1 \oplus P_2 \end{aligned}$$

(b)

Voraussetzung:

Da $\underline{P_1} \leq \underline{Q}$, gibt es eine Funktion $f_1 : \mathbb{N} \to \mathbb{N}$, so dass gilt:

$$n \in P_1 \Leftrightarrow f_1(n) \in Q \quad \forall n \in \mathbb{N}$$

Ebenso gibt es, da $\underline{P_2} \leq \underline{Q}$, eine Funktion $f_2 : \mathbb{N} \to \mathbb{N}$, so dass gilt:

$$n \in P_2 \Leftrightarrow f_2(n) \in Q \quad \forall n \in \mathbb{N}$$

Gesucht ist eine Funktion $g : \mathbb{N} \to \mathbb{N}$, so dass gilt:

$$m \in P_1 \oplus P_2 \Leftrightarrow g(m) \in Q \quad \forall m \in \mathbb{N}$$

Folgende Funktion erfüllt dies:

$$g(m) := \begin{cases} f_1(\frac{m}{2}) & \text{falls } m \text{ gerade} \\ f_2(\frac{m-1}{2}) & \text{sonst} \end{cases}$$

Prüfe $m \in P_1 \oplus P_2 \Leftrightarrow g(m) \in Q$

☞ 1. Fall (m gerade)

$$\begin{aligned} m \in P_1 \oplus P_2 &\Leftrightarrow m \in \{2n \mid n \in P_1\} \cup \{2n+1 \mid n \in P_2\} \\ &\Leftrightarrow m \in \{2n \mid n \in P_1\} \quad \text{, da } m \text{ gerade} \\ &\Leftrightarrow \frac{m}{2} \in P_1 \\ &\Leftrightarrow f_1(\frac{m}{2}) \in Q \end{aligned}$$

☞ 2. Fall (m ungerade)

$$\begin{aligned} m \in P_1 \oplus P_2 &\Leftrightarrow m \in \{2n \mid n \in P_1\} \cup \{2n+1 \mid n \in P_2\} \\ &\Leftrightarrow m \in \{2n+1 \mid n \in P_2\} \quad \text{, da } m \text{ ungerade} \\ &\Leftrightarrow \frac{m-1}{2} \in P_2 \\ &\Leftrightarrow f_2(\frac{m-1}{2}) \in Q \end{aligned}$$

Aufgabe 22

(a) $X = (bbb, abb); Y = (bb, babbb)$

$i_1 = 1; i_2 = 2; i_3 = 1 \to bbbabbbbb$

(b) $X = (a, bba, aab); Y = (ba, aaa, ba)$

Die ersten Buchstaben aller Paare X_i, Y_i mit $i \in \{1, 2, 3\}$ kollidieren.

(c) $X = (a, aab, baaa); Y = (aa, bb, a)$

$i_1 = 1; i_2 = 1; i_3 = 2; i_4 = 3; i_5 = 1; i_6 = 1 \rightarrow aaaabbaaaaa$

(d) $X = (bb, baa, ab, aa); Y = (ab, ba, aa, aba)$

☞ $i_1 = 2$, da alle anderen Paare X_i, Y_i mit $i \in \{1, 3, 4\}$ kollidieren.

- $i_2 \stackrel{?}{=} 1 \rightarrow$ Kollision bei i_3
- $i_2 \stackrel{?}{=} 2 \rightarrow$ Kollision!
- $i_2 \stackrel{?}{=} 3 \rightarrow$ Kollision bei i_3
- $i_2 \stackrel{?}{=} 4 \rightarrow$ Kollision!

Aufgabe 23 Teil 1

(a) Idee:

☞ Eine Zeilenparkettierung, d.h. eine unendliche Folge $(m_0, m_1)(m_1, m_2)(m_2, m_3)\ldots$ mit endlichem $D = \{d_1, \ldots, d_k\}$ von Dominotypen ist nur realisierbar, wenn es eine Folge $(m_i, m_{i+1})(m_{i+1}, m_{i+2})\ldots(m_j, m_{j+1})$ mit $m_{j+1} = m_i$ gibt. Diese kann dann unendlich oft wiederholt werden (da es von jedem Dominotyp beliebig d_i viele zur Verfügung stehen), so dass eine Zeilenparkettierung entsteht.

☞ Wenn es einen Zyklus mit den k Dominotypn aus D verwirklichbar ist, dann hat er höchstens die Länge $k + 1$ (da nur k Typen zur Verfügung stehen).

(b) Algorithmus, der das Problem entscheidet:

1. Bilde alle Folgen der Länge $k + 1$ (k^{k+1} viele)
2. Prüfe alle Folgen, ob sie gültige Folgen, d.h. ob sie Präfix einer Zeilenparkettierung sind. Gibt es eine solche Folge, so terminiere mit 1, sonst mit 0, da bei nur k Dominotypen in einer Folge der Länge $k + 1$ mindestens ein Dominotyp zweimal vorkommt!

Aufgabe 23 Teil 2

$X = (x_1, \ldots, x_k); Y = (y_1, \ldots, y_k)$ mit $x_i; y_i \in \{a\}^*$

(a) zz. PCP(1) ist entscheidbar

Folgender Algorithmus entscheidet das Problem:

1. zu X und Y bilde Folge $D = (d_1, \ldots, d_k)$ mit $d_i = |\,x_i\,| - |\,y_i\,|$
2. Prüfe dann:

   ☞ 1.Fall: $\exists\, d_i \in D : d_i = 0 \Rightarrow$ terminiere mit 1, da i Lösung des Problems.

   ☞ 2.Fall: $\forall\, d_i \in D : d_i > 0 \;\vee\; \forall\, d_i \in D : d_i < 0 \Rightarrow$ terminiere mit 0.

   ☞ 3.Fall: sonst terminiere mit Ausgabe 1, da die Lösung folgende Gestalt hat:

$$\exists\, i, j : d_i > 0 \;\wedge\; d_j < 0 \Rightarrow \text{ Indexfolge: } \underbrace{i, \ldots, i}_{|\,d_j\,|}, \underbrace{j, \ldots, j}_{|\,d_i\,|}$$

Serie 9

Aufgabe 24

$$\boxed{O(g) = \{f \mid \exists\, c > 0, \exists\, n_0 > 0 \;:\; \forall\, n \geq n_0 \;:\; f(n) \leq c \cdot g(n)\}}$$

(a)

zz. $6n^2 + 12n + 10 \in O(n^2)$

$$\Rightarrow \text{zz. } 6n^2 + 12n + 10 \leq c \cdot n^2 \quad \text{für } n \geq n_0$$

Wähle $c = 7$.

$$\begin{aligned}
&\Rightarrow -n^2 + 12n + 10 \leq 0 \\
&\Leftrightarrow n^2 - 12n - 10 \geq 0 \\
&\Leftrightarrow n \geq 6 + \sqrt{46} \;\vee\; n \leq 6 - \sqrt{46} \\
&\Leftrightarrow n \gtrsim 12{,}78 \;\vee\; n \lesssim -0{,}78 \\
&\Rightarrow n^2 - 12n - 10 \geq 0 \quad \forall\, n \geq n_0 \text{ mit } n_0 = 13
\end{aligned}$$

(b)

zz. $2^{n+10} \in O(2^n)$

$$\Rightarrow \text{zz. } 2^{n+10} \leq c \cdot 2^n \quad \text{für } n \geq n_0$$

Wähle $c = 2^{10} = 1024$.

$$\begin{aligned}
&\Rightarrow 2^{n+10} \leq 2^{10} \cdot 2^n \\
&\Leftrightarrow 2^{n+10} \leq 2^{n+10} \\
&\Leftrightarrow 0 \leq 0 \quad \forall\, n \geq n_0 \text{ mit } n_0 = 0
\end{aligned}$$

(c)

$n^2 \overset{?}{\in} O(n \cdot \log^2 n)$

Nein, da n^2 stärker ansteigt als $n \cdot \log^2 n$, da für jedes c ab einer gewissen Stelle n_0 gilt, dass $n^2 > c \cdot n \cdot \log^2 n$, da n stärker wächst als $\log^2 n$.

(d)

$n \cdot \log n \overset{?}{\in} O(n^2)$

Ja, da $n \in O(n) \;\wedge\; \log n \in O(n)$.

(e)

$3^n \overset{?}{\in} 2^O(n)$

zz. $\exists\, c > 0\; \exists\, n_0 > 0 \;:\; \forall\, n \geq n_0 \;:\; 3^n \leq 2^{c \cdot n}$

$$\Rightarrow 3^n \leq 2^{c \cdot n} \quad \Leftrightarrow \quad (2^{\log_2 3})^n \leq 2^{c \cdot n} \quad \Leftrightarrow \quad 2^{n \cdot \log_2 3} \leq 2^{c \cdot n}$$

Mit $c = \log_2 3$ und $n_0 = 1$ gilt $\forall\, n \geq n_0 \;:\; 3^n \leq 2^{c \cdot n} \Rightarrow 3^n \in 2^{O(n)}$

Aufgabe 25

Folgende Turingmaschine $M = (Q, \Sigma, \Gamma, q_0, q_s, \delta)$ verdoppelt die Eingabe:

1. Phase

   ☞ Gehe bei Beibehaltung der Inschrift zum ersten Blank und ersetze dies durch \$.

   $$q_0 a_1 \ldots a_n \sqcup \underset{M}{\vdash^*} a_1 \ldots a_n q_0 \sqcup \underset{M}{\vdash} a_1 \ldots a_n q_1 \$ \sqcup$$

   ☞ Gehe dann zurück zum Anfang des Wortes:

   $$a_1 \ldots a_n q_1 \$ \sqcup \underset{M}{\vdash^*} q_1 \sqcup a_1 \ldots a_n \$ \underset{M}{\vdash} q_2 a_1 \ldots a_n \$$$

2. Phase

   ☞ Merke den Buchstaben des Arbeitsfeldes in einem entsprechenden Zustand, falls dieser Buchstabe $\neq \$$ (zu merkender Buchstabe $a_i \rightarrow$ gehe in Zustand q_{a_i}).

   ☞ Markiere dabei den gelesenen Buchstaben durch Unterstrich und gehe nach rechts zum 1. Blank und schreibe dort den entsprechenden Buchstaben:

   $$q_2 a_1 \ldots a_n \$ \sqcup \underset{M}{\vdash} \underline{a_1} q_{a_1} a_2 \ldots a_n \$ \sqcup \underset{M}{\vdash^*} \underline{a_1} a_2 \ldots a_n q_3 \$ a_1 \sqcup$$

   ☞ Ist der aktuelle Buchstabe \$, so überschreibe ihn mit $\sqcup$ und fahre mit Phase 4 fort.

3. Phase

   ☞ Laufe zurück zum nächsten zu kopierenden Buchstaben, fahre fort mit Phase 2.

4. Phase

   ☞ Entferne alle Unterstriche und gehe zum Anfang des Wortes.

Laufzeitanalyse:

1. Phase:	$n+1+n+1$	Schritte :	$\in O(n)$	
2. Phase:	$n+2$	Schritte :	$\in O(n)$ - wird n-mal durchlaufen	$\Rightarrow \in O(n^2)$
3. Phase:	$n+2$	Schritte :	$\in O(n)$ - wird n-mal durchlaufen	
4. Phase:	$n+1$	schritte :	$\in O(n)$	

Aufgabe 26

Sei $|\, w \,| = n$. Ist eine Wortfunktion f_i durch eine polynomzeitbeschränkte Turingmaschine M_i mit Polynomzeitschranke $p_i(n)$ berechenbar, dann kann sich durch diese Funktion die Eingabe höchstens um $p_i(n)$ Zeichen verlängern. $\rightarrow$ Ist $|\, w \,| = n$ die Eingabe von M_i dann hat die Ausgabe von M_i eine Länge $\leq n + p_i(n)$.

w	$\longrightarrow$	$f_1(w)$	$\longrightarrow$	$f_2(f_1(w))$
Eingabe M_1 Länge $\|\, w \,\| = n$	Wortfunktion f_1 Schritte $\leq p_1(n)$	Ausgabe M_1, Eingabe M_2 Länge $\leq n + p_1(n)$	Wortfunktion f_2 Schritte $\leq p_2(n + p_1(n))$	Ausgabe $M2$

$\Rightarrow$ Schritte gesamt: $p_1(n) + p_2(n + p_1(n)) = q(n)$ mit $q(n)$ offensichtlich ein Polynom.

Serie 10

Aufgabe 27

(a)

Sei $d(n)$ das Eingabewort, dann entscheidet Z folgende polynomzeitbeschränkte nichtdeterministische Turingmaschine (NTM) M:

1. Phase

   ☞ Schreibe deterministisch hinter 1. $\sqcup$ nach dem Eingabewort $|\,d(n)\,|$-mal $|$:

   $$d(n) \mapsto^* d(n) \sqcup |^{|\,d(n)\,|}$$

2. Phase

   ☞ Ersetze $|^{|\,d(n)\,|}$ nichtdeterministisch durch eine Dezimalzahl $d(n_1)$ höchstens der Länge von $|\,d(n)\,|$. Alle dabei nicht benötigten $|$ werden durch $\sqcup$ ersetzt:

   $$d(n) \sqcup |^{|\,d(n)\,|} \mapsto^* d(n) \sqcup d(n_1) \sqcup \sqcup \sqcup \ldots$$

3. Phase

   ☞ Schreibe nichtdeterministisch $|\,d(n)\,|$ viele $|$ hinter 1. $\sqcup$ nach $d(n_1)$:

   $$d(n) \sqcup d(n_1) \sqcup \mapsto^* d(n) \sqcup d(n_1) \sqcup |^{|\,d(n)\,|}$$

4. Phase

   ☞ Ersetze diese $|^{|\,d(n)\,|}$ wiederum nichtdeterministisch durch eine Dezimalzahl $d(n_2)$. Analog zu Phase 2 hat $d(n_2)$ höchstens Länge $|\,d(n)\,|$, alle überflüssigen $|$ werden durch $\sqcup$ ersetzt:

   $$d(n) \sqcup d(n_1) \sqcup |^{|\,d(n)\,|} \mapsto^* d(n) \sqcup d(n_1) \sqcup d(n_2)$$

5. Phase

   ☞ Multipliziere in Polynomzeit $d(n_1)$ und $d(n_2)$ miteinander, schreibe das Ergebnis links vom Eingabewert $d(n)$ auf das Band:

   $$d(n) \sqcup d(n_1) \sqcup d(n_2) \mapsto^* d(n_1 \cdot n_2) \sqcup d(n) \sqcup d(n_1) \sqcup d(n_2)$$

6. Phase

   ☞ Vergleiche $d(n_1 \cdot n_2)$ mit $d(n)$. Terminiere mit "ja", falls $d(n_1 \cdot n_2) = d(n)$, andernfalls terminiere mit "nein".

$\Rightarrow$ Alle Phasen sind polynomzeit-berechenbar. $\Rightarrow$ M entscheidet $Z \Rightarrow Z \in NP$.

Anmerkungen zur polynomzeitbeschränkten NTM:

Eine polynomzeitbeschränkte nichtdeterministische Turingmaschine besteht nur aus polynomzeit-berechenbaren deterministischen Phasen, die Phasen einer normalen Turingmaschine entsprechen und polynomzeit-berechenbaren nichtdeterministischen Phasen. In diesen Phasen werden parallel verschiedene "Programmpfade" beschritten. In obiger Aufgabe werden z.B. in Phase 2 parallel alle möglichen verschiedenen Werte für $d(n_1)$ bearbeitet (sind nur endlich viele, da die Länge $\leq |\,d(n)\,|$), terminiert dann ein "Pfad" (sprich für eine Kombination von $d(n_1)$ und $d(n_2)$) der NTM mit "ja", dann ist auch das Gesamtergebnis "ja".

(b)

Anmerkungen zum polynomial größenbeschränkten Suchproblem:

☞
gegeben:	$x \in$ Instanzenmenge I
Frage:	$\exists\, y$ mit $|\,y\,| \leq q(|\,x\,|)$, so dass $(x, y) \in R$ mit R entscheidbar in Polynomzeit und q Polynom.

☞ $L \subseteq \Sigma^*$ ist polynomial größenbeschränktes Suchproblem $\Leftrightarrow$ eine Relation $R \subseteq \Sigma^* \times \Gamma^*$ polynomzeit-entscheidbar ist und q ein Polynom, so dass gilt:

$$u \in L \Leftrightarrow \exists v \;:\; (|\,v\,| \leq q(|\,u\,|) \;\wedge\; (u, v) \in R)$$

zur Aufgabe:

gegeben: $d(n) \in$ Menge aller Dezimalzahlen

Frage: $\exists\, d(n_1)\#d(n_2)$ mit $|\, d(n_1)\#d(n_2)\,| \leq \underbrace{|\, d(n)\,| + 2}_{\text{Polynom } q} \wedge \underbrace{n_1 \cdot n_2 = n \wedge n_1 \neq 1 \wedge n_2 \neq 1 \wedge n \geq 2}_{\text{polynomzeit-entscheidbare Relation } R}$

$$\Rightarrow R = \{(d(n), d(n_1)\#d(n_2)) \mid n_1, n_2, n \in \mathbb{N},\ n \geq 2,\ n_1, n_2 \neq 1,\ n = n_1 \cdot n_2\}$$

Aufgabe 28

(a)

$K \cdot L$ ist ein polynomial größenbeschränktes Suchproblem.

gegeben: $w \in \{0,1\}^*$

Frage: $\exists\, u\#v\ :\ (|\, u\#v\,| \leq |\, w\,| + 1\ \wedge\ (w, u\#v) \in R)$

mit $R = \{(w, u\#v) \mid \underbrace{u \in K,\ v \in L}_{\text{(nach Voraussetzung in Polynomzeit)}},\ \underbrace{w = uv}_{\text{(polynomzeit entscheidbar)}}\}$

(b)

Nach Voraussetzung existieren Turingmaschinen M_K und M_L, die in Polynomzeit K bzw. L entscheiden. Sei $w = b_1 \dots b_n$ Eingabewort auf 1. Band, so entscheidet folgende 3-Band-Turingmaschine $K \cdot L$:

1. Phase

   ☞ Schreibe hinter w alle möglichen Zerlegungen $u\#v$ von w jeweils durch $\sqcup$ getrennt.

   $$b_1 \dots b_n \mapsto^* b_1 \dots b_n \sqcup \#b_1 \dots b_n \sqcup b_1\#b_2 \dots b_n \sqcup \dots \sqcup b_1 \dots b_n\# \sqcup$$

2. Phase

   ☞ Kopiere 1. u auf Band 2 und 1. v auf Band 3.

3. Phase

   ☞ Prüfe auf Band 2 mit Hilfe von M_K, ob $u \in K$.
   Prüfe auf Band 3 mit Hilfe von M_L, ob $v \in L$.

   - 1. Fall:
     Liefern sowohl M_K als auch M_L Antwort "ja"$\Rightarrow$ terminiere mit "ja".
   - 2. Fall:
     sonst lösche Band 2 und 3. Betrachte dann nächste Zerlegung $u\#v$. (vgl. Phase 2)

Aufgabe 29

zu $m \times n$-Matrix M und $k \geq 1$ bilde Inputwort wie folgt:

$w =$ 1.Zeile#2.Zeile#$\dots n$.Zeile##k $\in$ Menge aller Inputwörter (alle Komb. aus $m \times n$ Matrix und $k \in \mathbb{N}$)

Frage zum polynomial größenbeschränktes Suchproblem:

$\exists\, v$ mit $|\, v\,| \leq |\, w\,|$, so dass v entsteht durch Ersetzen von
k #'s durch \$ und die so markierten Zeilen überdecken M (in Polynomzeit berechenbar)

Serie 11

Aufgabe 30

(a) $P = NP \Rightarrow L_0$ ist NP-vollständig

zu zeigen:

1. $L_0 \in NP$ $\surd$, da $L_0 \in P$ und nach Voraussetzung $P = NP$
2. $\forall\, L \in NP \;:\; L \leq pL_0$

zu 2:

Es gilt $L \leq pL_0$, falls eine polynomzeitberechenbare Funktion f existiert:

$$f \;:\; \Gamma^* \to \Sigma^* \quad \text{mit} \quad w \in L \Leftrightarrow f(w) \in L_0 \tag{30.1}$$

$L \in NP$ sei gegeben. Da nach Voraussetzung $P = NP$ gilt $L \in P$. Somit existiert polynomzeitbeschränkte Turingmaschine M_L, die L entscheidet.
Wähle $w_0 \in L_0$ und $w_1 \notin L_0$ (existiert da $L_0 \neq \emptyset$ und da $L_0 \neq \Sigma^*$). Folgende Transition f erfüllt dann Gleichung 30.1:

Zu w bestimme mit Hilfe von M_L, ob $w \notin L$ oder $w \in L$.

$$f(w) = \begin{cases} w_0 & \text{,falls } w \in L \\ w_1 & \text{,falls } w \notin L \end{cases}$$

Somit klar, dass $w \in L \Leftrightarrow f(w) \in L_0$, also Polynomzeitreduktion gefunden.

(b) L_0 ist NP-vollständig $\Rightarrow P = NP$

Da L_0 NP-vollständig ist, gilt:

$$\text{für alle } L \in NP \;:\; L \leq pL_0$$

Nach Aufgabenstellung ist auch $L_0 \in P$. Damit folgt nach Vorlesung, dass auch $L \in P$.

Aufgabe 31

gegeben ein aussagenlogischer Ausdruck in DNF:

$$\beta = c_1 \vee \ldots \vee c_n \quad \text{mit} \quad c_i = l_{i_1} \wedge l_{i_2} \wedge \ldots \wedge l_{i_{k_i}} \tag{31.1}$$

zz. Erfüllbarkeitsproblem für aussagenlogische Ausdrücke in DNF ist in Polynomzeit lösbar

Idee:

Ein aussagenlogischer Ausdruck in DNF ist erfüllbar, wenn er eine Klausel enthält, die erfüllbar ist. Eine Klausel der Form $l_1 \wedge \ldots \wedge l_k$ ist nicht erfüllbar, wenn in ihr eine Variable x_j sowohl negiert als auch nicht negiert vorkommt $(x_j \wedge \neg x_j)$.

Algorithmus:

Eingabe: β (vgl. Definition 31.1)

1. Wähle die nächste Klausel c_i (zu Beginn 1. Klausel c_1). Falls keine Klausel mehr zu wählen ist, terminiere mit Ausgabe 0.
2. Prüfe in c_i, ob es eine Variable x_j gibt, die in c_i sowohl negiert als auch nicht negiert vorkommt.
   - ☞ 1. Fall: Gibt es eine solche Variable, so fahre mit 1 fort.
   - ☞ 2. Fall: Gibt es keine solche Vairiable, so terminiere mit Ausgabe 1.

Aufgabe 32

(a)

gegeben: $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, $l \in \mathbb{N}$

Folgende Sprache L' kodiert das PCP'-Problem:

$$L' = \left\{ x_1\# \ldots \#x_n\$y_1\# \ldots \#y_n\$\underbrace{||\ldots|}_{l\text{-mal}} \;\middle|\; \exists\, i_1, \ldots, i_m, m \leq l \;:\; x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m} \right\} \tag{32.1}$$

(b)

L' ist polynomial größenbeschränktes Suchproblem:

$$u \in L' \Leftrightarrow \exists\, v \;:\; \big(|\,v\,| \leq q(|\,u\,|)\ \wedge\ \underbrace{(v \text{ ist Lösung von } u)\ \wedge\ (u \text{ hat obige Form, vgl. Gleichung 32.1})}_{\text{in Polynomzeit entscheidbar}}\big)$$

v = Indexfolge hat dabei folgende Form:

$$|^{i_1}\#|^{i_2}\#\ldots\#|^{i_m} \quad \text{mit } m \leq l$$

Für jedes i_j gilt: $i_j \leq n$. Somit gilt:

$$|\,v\,| \leq \underbrace{(n + \overbrace{1}^{\text{wg. }\#})}_{\leq |\,u\,|} \cdot \underbrace{l}_{\leq |\,u\,|} \quad \Rightarrow \quad |\,v\,| \leq |\,u\,|^2$$

Serie 12

Aufgabe 33

gegeben:

KNF-Formel $\beta = \bigwedge_{i=1}^{m} (l_{i_1} \vee l_{i_2} \vee l_{i_3})$ über den Variablen $x1, \ldots, x_n$

gesucht:

Erfüllende Belegung von $x1, \ldots, x_n$ für β, d.h. pro Klausel muß ein Literal existieren, das wahr gesetzt werden kann.

Algorithmus:

Folgender Algorithmus findet, falls vorhanden, eine erfüllende Belegung oder gibt $\emptyset$ zurück, falls diese nicht existiert:

Eingabe: KNF-Formel mit 3 Literalen pro Klausel

/* Algorithmus im Pseudo-Code */

$L := \emptyset$
return belegung(1)

/* Prozedur belegung */

```
belegung(Klausel i)={
    if(i = m + 1) then return L;
    if(kein Konflikt zwischen L und l_{i_1}) then {
        L := L ∪ {l_{i_1}};
        if(belegung(i + 1)≠ ∅) then return L; else L := L\{l_{i_4}};
    };
    if(kein Konflikt zwischen L und l_{i_2})) then {
        L := L ∪ {l_{i_2}};
        if(belegung(i + 1)≠ ∅) then return L; else L := L\{l_{i_2}};
    };
    if(kein Konflikt zwischen L und l_{i_3})) then {
        L := L ∪ {l_{i_3}};
        if(belegung(i + 1)≠ ∅) then return L; else L := L\{l_{i_3}};
    };
return ∅;
}
```

Erläuterungen:

Der Algorithmus fängt bei der ersten Klausel an und setzt das erste Literal auf wahr (durch Aufnahme in die erfüllende Belegung L), geht dann zur zweiten Klausel und versucht dort wieder das erste Literal auf wahr zu setzen. Geht dies nicht, da es aufgrund der vorherigen Aktion bereits negativ gesetzt ist (gibt es also einen Konflikt zwischen L und l_{i_k}, so wird versucht das nächste Literal in der Klausel positiv zu setzen. Kann kein Literal in einer Klausel poitiv gesetzt werden, wird wieder zur vorhergehenden Klausel zurückgegangen und dort versucht ein anderes Literal positiv zu setzen ($\rightarrow$ Backtracking).
Wenn alle Klauseln durchlaufen sind und jeweils ein Literal positiv gesetzt worden ist, ist L die erfüllende Belegung. Andernfalls, wenn das Backtracking dazu führt, dass selbst das Setzen des letzten Literals in der ersten Klausel nicht zum Erfolg führt, wird $L = \emptyset$ zurückübergeben ($\rightarrow$ es gibt keine erfüllende Belegung).

Aufgabe 34

gesucht ist also eine $\log(n)$-platzbeschränkte deterministische Offline-Turingmaschine, die L entscheidet, d.h. sie benutzt bei Eingabelänge n höchstens $O(\log n)$ viele Felder auf dem Arbeitsband.

Folgende Offline-Turingmaschine leistet dies:

Eingabe: $x \in \{0, 1, c\}^*$ - Bsp.: $x = \cent\, 101c101\$$

1. Phase
   - ☞ Durchlaufe x bis zum ersten c und erhöhe dabei für jede gelesene 1 oder 0 einen Binärzähler b_1 auf Arbeitsband um 1.
   - ☞ Sollte bis zum \$ bzw. ersten $\sqcup$ kein c gelesen werden, so terminiere mit 0.
2. Phase
   - ☞ Schreibe, durch $\sqcup$ getrennt, einen zweiten Binärzähler b_2 hinter b_1.
     b_2 habe dabei $\mid b_1 \mid$-viele 0en : $b_1 \sqcup \underbrace{0 \ldots 0}_{\mid b_1 \mid\text{-mal}}$
   - ☞ Gehe auf Eingabeband zurück zum ersten Buchstaben des Eingabewortes.
3. Phase
   - ☞ Merke gelesenen Buchstaben von x in Zustand.
   - ☞ Gehe in diesem Zustand nach rechts auf Eingabeband. Erhöhe pro gelesenen Buchstaben jeweils den Zähler b_2 um 1 und verringere b_1 um 1.
   - ☞ Fahre mit Phase 4 fort, wenn $b_1 = 0 \ldots 0$

4. Phase

   ☞ Gehe auf Eingabefeld ein Feld nach rechts und überprüfe, ob der im Zustand gespeicherte Buchstabe dort steht.

   - 1. Fall:
     gespeicherter Buchstabe: c
     gelesener Buchstabe: \$
     $\Rightarrow$ terminiere mit 1
   - 2. Fall:
     gespeicherter Buchstabe: c
     gelesener Buchstabe: nicht \$
     $\Rightarrow$ terminiere mit 0
   - 3. Fall:
     gespeicherter Buchstabe $\triangleq$ gelesener Buchstabe
     $\Rightarrow$ weiter mit Phase 5
   - 4. Fall:
     sonst $\Rightarrow$ terminiere mit 0

5. Phase

   ☞ Gehe auf Eingabeband nach links, erhöhe dabei pro Buchstabe jeweils Zähler b_1 um 1 und verringere b_2 um 1 (bis $b_2 = 0 \ldots 0$)

   ☞ Fahre mit Phase 3 fort.

Der Platzbedarf ist, da es nur zwei Binärzähler b_1 und b_2 gibt, nur $O(\log n) \Rightarrow L \in DLOG$.

Aufgabe 35

gesucht ist eine polynomzeitberechenbare Funktion:

$$f : I_C \rightarrow I_I \quad \text{mit Instanzenmengen der Probleme } I_C = I_I = (\text{Menge aller Graphen}) \times \mathbb{N}$$

und mit

$$x \in CLIQUE \Leftrightarrow f(x) \in INDEPENDENT\ SET \tag{35.1}$$

f liefert bei Eingabe $G(V, E)$ und $k \in \mathbb{N}$:

$$f((G, k)) = ((V, V \times V \setminus E \cup \{(x, x) \mid x \in V\}), k)$$

Offensichtlich gilt Gleichung 35.1 und f ist polynomzeitberechenbar (eigtl. noch zu zeigen).

Serie 13

Aufgabe 36

keine Lösung vorhanden.

Aufgabe 37

keine Lösung vorhanden.

Aufgabe 38

keine Lösung vorhanden.