**2. Klausur zur Vorlesung**
**Theoretische Grundlagen der Informatik**
**Wintersemester 2022/2023**

# Lösung!

- Bringen Sie den Aufkleber mit Ihrer Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihrer Matrikelnummer.
- Schreiben Sie die Lösungen auf die Aufgabenblätter und Rückseiten. Am Ende der Klausur sind zusätzliche Leerseiten. Fordern Sie zusätzliches Papier bitte nur an, falls Sie den gesamten Platz aufgebraucht haben.
- Die Tackernadel darf nicht gelöst werden.
- Als Hilfsmittel ist ein beschriebenes A4-Papier erlaubt.
- Einlesezeit: 15 min
  Bearbeitungszeit: 2 h

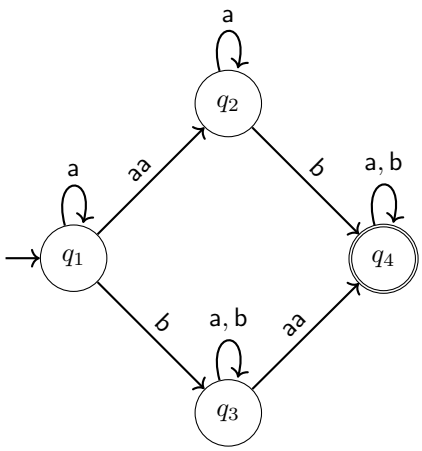
| | Mögliche Punkte | | | | | | Erreichte Punkte | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | a | b | c | d | e | Σ | a | b | c | d | e | Σ |
| Aufg. 1 | 2 | 2 | 3 | 3 | – | 10 | | | | | – | |
| Aufg. 2 | 3 | 3 | 2 | – | – | 8 | | | | – | – | |
| Aufg. 3 | 3 | 1 | 2 | 5 | 3 | 14 | | | | | | |
| Aufg. 4 | 1 | 1 | 6 | – | – | 8 | | | | – | – | |
| Aufg. 5 | 1 | 2 | 1 | 4 | 3 | 11 | | | | | | |
| Aufg. 6 | 1 | 5 | 3 | – | – | 9 | | | | – | – | |
| Σ | | | | | | 60 | | | | | | |

## **Problem 1:** Block-NEA $2 + 2 + 3 + 3 = 10$ Punkte

Wir definieren einen *k-Block-NEA* wie einen NEA, mit der Änderung, dass pro Schritt bis zu $k$ Zeichen gelesen werden dürfen. Ein $k$-Block-NEA hat also auch eine Zustandsmenge $Q$, einen Startzustand $s$, Endzustände $F \subseteq Q$ und ein Alphabet $\Sigma$. Die Übergangsfunktion $\delta$ kann bei jedem Übergang bis zu $k$ Zeichen lesen.

Formal hat ein Übergang also die Form $\delta(q, w) = p$, wobei $p$ und $q$ Zustände sind und $w \in \Sigma^*$ ein Wort der Länge $|w| \leqslant k$ ist. Ein Wort $w$ wird genau dann akzeptiert, wenn es eine Folge von Übergängen durch Zustände $q_0, q_1, \ldots, q_\ell$ mit $\delta(q_i, w_{i+1}) = q_{i+1}$ gibt, wobei $q_0 = s$ der Startzustand ist, $q_\ell \in F$ akzeptierend und $w = w_1 \cdot w_2 \cdot \ldots \cdot w_\ell$ die Konkatenation der jeweils gelesenen Wörter ist.

(a) Welche Sprache erkennt der folgende 2-Block-NEA?




(b) Geben Sie einen 3-Block-NEA an, der genau die Wörter über dem Alphabet $\{\mathsf{a}, \mathsf{b}\}$ erkennt, in denen das Teilwort aab eine gerade Anzahl oft vorkommt.

(c) Zeigen Sie, dass $k$-Block-NEAs gleich mächtig wie NEAs sind.

Betrachten wir nun nur Sprachen, deren Wörter ein Vielfaches von $k$ lang sind. Wir interessieren uns für $k$-Block-NEAs, bei denen in jedem Schritt genau $k$ Zeichen gelesen werden, diese Automaten nennen wir *vollständig*.

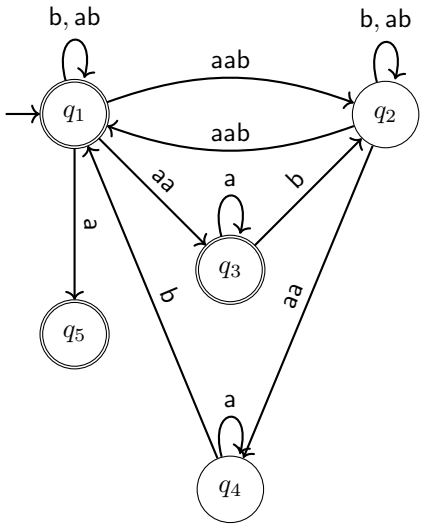
(d) Sei $\mathcal{A}$ ein $k$-Block-NEA. Beschreiben Sie, wie ein vollständiger $k$-Block-NEA $\mathcal{A}'$, der die gleiche Sprache wie $\mathcal{A}$ erkennt, konstruiert werden kann.

*Hinweis: Es ist hilfreich, $\mathcal{A}$ zuerst in einen NEA umzuwandeln. Hierzu dürfen Sie (c) verwenden, auch wenn Sie die Teilaufgabe nicht bearbeitet haben. Konstruieren Sie dann daraus einen vollständigen $k$-Block-NEA.*

**Lösung:**

(a) Der Automat erkennt die Sprache aus allen Wörtern, die das Teilwort aa und mindestens ein b beinhalten.

(b)




(c) Jeder NEA ist ein $k$-Block-NEA, der immer nur ein Zeichen liest, also sind NEAs nicht mächtiger. Ein beliebiger $k$-Block-NEA kann in einen NEA, der die gleiche Sprache erkennt, umgewandelt werden, indem jeder Übergang, der mehrere Zeichen liest, in eine Folge von Übergängen mit nur einem Zeichen ersetzt wird (dabei werden neue Zustände eingefügt). Somit sind NEAs und $k$-Block-NEAs gleich mächtig.

(d) Wandle $\mathcal{A}$ in einen NEA $\bar{A}$ um, wie in der obigen Teilaufgabe beschrieben. Der vollständige $k$-Block-NEA $A'$, der die gleiche Sprache entscheidet, hat die gleiche Zustandsmenge wie $\bar{A}$. Füge für jeden Pfad der Länge $k$ in $\bar{A}$ zwischen Zuständen $p$ und $q$ einen Übergang von $p$ nach $q$ mit dem korrespondierenden Wort der Länge $k$ in $\mathcal{A}'$ ein. Formal bedeutet das, dass für jede Folge von $k$ Übergängen $(p_1, z_1) \to p_2, (p_2, z_2) \to p_3, \ldots, (p_k, z_k) \to q_{k+1}, \quad p_i \in Q(\bar{A}), z_i \in \Sigma$ in $\bar{A}$ ein Übergang $(p_1, z_1 \ldots z_k) \to q_{k+1}$ in $\mathcal{A}'$ eingefügt wird.

*Alternative Lösung:* Wir können Übergänge, die weniger als $k$ Zeichen lesen, ähnlich wie $\varepsilon$-Übergänge in NEAs betrachten und entsprechend ersetzen. Nennen wir Übergänge, die weniger als $k$ Zeichen lesen *unvollständig* (andere heißen *vollständig*). Jeden unvollständigen Übergang $\delta(q, w) = p$ von $\mathcal{A}$ können wir zu einer Menge vollständiger Übergänge erweitern, indem wir von $p$ alle Wörter $w'$, so dass $|w| + |w'| = k$ gilt, betrachten und $\delta(q, w)$ zu $\bar{\delta}(q, ww') = \delta(p, w')$ erweitern. Wenn dabei ein Übergang $\delta(p, w'u) = r$ nicht vollständig betrachtet werden kann, da die Menge der gelesenen Zeichen $w'u$ länger als $|w'|$ ist, fügen wir jeweils einen neuen Hilfszustand $h$ ein, der repräsentiert, dass der Übergang von $p$ nach $r$ nach Lesen von $w'$ unterbrochen wurde. Von $h$ aus führt Lesen der restlichen Zeichen $u$ in Zustand $r$ (andere Eingaben führen in den impliziten Müllzustand). Der Prozess muss auch für entstandene Hilfszustände wiederholt werden, bis keine unvollständigen Übergänge mehr existieren. Dieser Vorgang ist endlich, weil für jeden ursprünglichen Übergang maximal ein Hilfszustand nach jedem gelesenen Zeichen eingefügt werden kann.

## **Problem 2:** Turingmaschinen 3 + 3 + 2 = 8 Punkte

Wir definieren die Sprache

$$L_{\text{xor}} = \{x\#y \mid x, y \in \{0,1\}^*, |x| = |y| \text{ und } x \oplus y = 1^{|x|}\},$$

wobei $\oplus$ bitweises XOR darstellt. Für zwei Bits $a$ und $b$ gilt: $a \oplus b = 1 \iff a \neq b$.

Beispielsweise sind die Wörter 110#001 und 1#0 in $L_{\text{xor}}$, aber 11#0 und 011#010 sind nicht in $L_{\text{xor}}$.

*Hinweis: Der Begriff* Turingmaschine *meint eine Turingmaschine wie in der Vorlesung definiert, insbesondere mit nur einem Band und nur einem Kopf.*

(a) Beschreiben Sie eine Turingmaschine, die $L_{\text{xor}}$ im $\mathcal{O}$-Kalkül möglichst effizient entscheidet. Geben Sie außerdem die Laufzeit im $\mathcal{O}$-Kalkül an.

Unser Ziel wird nun sein zu zeigen, dass es keine Turingmaschine gibt, die $L_{\text{xor}}$ schneller als in quadratischer Zeit entscheidet. Dazu definieren wir die Sprache

$$L_{\text{dup}} = \{w\#w \mid w \in \{0,1\}^*\} .$$

Für diese Sprache ist Folgendes bekannt: Es gibt keine Turingmaschine, die $L_{\text{dup}}$ schneller als in quadratischer Zeit entscheidet. Formal bedeutet das, dass jede Turingmaschine, die $L_{\text{dup}}$ entscheidet, im Worst Case $\Omega(n^2)$ Schritte benötigt, wobei $n$ die Eingabelänge ist.

(b) Geben Sie eine Transformation $f$ von $L_{\text{dup}}$ nach $L_{\text{xor}}$ an, die von einer Turingmaschine in Linearzeit berechenbar ist. Für alle Wörter $w$ soll also

$$w \in L_{\text{dup}} \iff f(w) \in L_{\text{xor}}$$

gelten. Beweisen Sie die Korrektheit und Laufzeit Ihrer Transformation $f$.

(c) Zeigen Sie, dass es keine Turingmaschine gibt, die $L_{\text{xor}}$ schneller als in quadratischer Zeit entscheidet.

**Lösung:**

(a) Falls die Eingabe nicht genau ein # enthält, lehnt die Turingmaschine ab.

Der Kopf bewegt sich zum ersten Zeichen in $x$, das eine 0 oder eine 1 ist, speichert das Bit im Zustand und überschreibt das Zeichen mit einem $X$. Dann bewegt sich der Kopf zum ersten Zeichen in $y$, das eine 0 oder eine 1 ist. Falls es kein solches Zeichen gibt, lehnt die Turingmaschine ab, da der zweite Teil zu kurz ist. Ansonsten vergleicht sie das Bit mit dem gespeicherten Bit. Sind diese gleich, lehnt die Turingmaschine direkt ab. Ansonsten löscht sie das Bit vom Band. Diese Schritte werden so oft wiederholt bis vor dem ersten # alle Zeichen mit $X$ überschrieben wurden. Falls nicht alle Zeichen nach # gelöscht sind, lehnt die Turingmaschine ab, da der zweite Teil dann zu lang ist. Ansonsten akzeptiert die Turingmaschine die Eingabe.

Für eine Eingabe der Länge $n$ bewegt sich der Kopf $n$ Mal hin und her, wobei die maximale Länge der Bandbeschriftung in $\mathcal{O}(n)$ ist. Insgesamt benötigt die Turingmaschine also $\mathcal{O}(n^2)$ Zeit.

(b) Sei $w$ ein gegebenes Wort. Wir konstruieren ein Wort $w'$, sodass $w \in L_{\text{dup}} \Leftrightarrow w' \in L_{\text{xor}}$. Falls $w$ nicht genau ein # enthält, definieren wir $w' \coloneqq \varepsilon$, womit $w' \notin L_{\text{xor}}$ gilt. Sonst hat $w$ die Form $x\#y$. Sei dann $w' = x\#\overline{y}$, wobei in $\overline{y}$ jedes Bit in $y$ negiert wird.

Falls $w \in L_{\text{dup}}$, ist $x = y$. Damit gilt $x \oplus \overline{y} = x \oplus \overline{x} = 1^{|x|}$, also $w' = x\#\overline{y} \in L_{\text{xor}}$.

Ist umgekehrt $w' \in L_{\text{xor}}$, so ist $w' \neq \varepsilon$, da $\varepsilon \notin L_{\text{xor}}$. Also enthält $w'$ genau ein # und hat die Form $w' = x\#\overline{y}$ mit $x = \overline{\overline{y}}$, also $x = y$. Damit ist $w = x\#y = x\#x$ und somit in $L_{\text{dup}}$.

Die Überprüfung, ob $w$ genau ein # enthält und das Negieren von $y$ sind jeweils in Linearzeit.

*Hinweis: Eine Transformation muss immer (wie z.B. auch bei der polynomiellen Transformation) alle möglichen Eingaben (hier: beliebige $w \in \Sigma^*$) abbilden. Es genügt also nicht nur Ja-Instanzen (hier: Wörter aus $L_{dup}$) abzubilden. Es ist nicht einmal möglich, in der gegebenen Zeit zu testen, ob es sich um eine Ja-Instanz handelt! Wenn hier also etwas nur für Wörter aus $L_{dup}$ getan wurde, so erfüllt das nicht die Aufgabenstellung. Merkhilfe: Denken Sie an die Anwendung einer Transformation, wie in Aufgabenteil (c): Gegeben ist ein beliebiges Wort $w$ und wir wollen mit Hilfe der Transformation einen Algorithmus bauen, der entscheidet, ob $w \in L_{dup}$. Es wäre also nicht sinnvoll, zu Beginn der Transformation zu testen, ob $w \in L_{dup}$. Wenn wir das könnten, dann bräuchten wir die Transformation nicht.*

(c) Angenommen, es gibt eine 1-Band Turingmaschine $M_{\text{xor}}$, die $L_{\text{xor}}$ in weniger als quadratischer Zeit entscheidet. Dann konstruieren wir eine 1-Band Turingmaschine $M_{\text{dup}}$, die $L_{\text{dup}}$ in weniger als quadratischer Zeit entscheidet: Eine Eingabe $w$ wird zunächst in $w'$ wie in Teilaufgabe (b) in linearer Zeit transformiert. Dann ist $|w'|$ linear in $|w|$. Auf $w'$ verhält sich $M_{\text{dup}}$ wie $M_{\text{xor}}$ und berechnet in subquadratischer Zeit, ob $w' \in L_{\text{xor}}$ gilt. Falls ja, akzeptiert $M_{\text{dup}}$. Sonst lehnt die Turingmaschine ab. Da die Transformation nur lineare Zeit benötigt, benötigt $M_{\text{dup}}$ im $\mathcal{O}$-Kalkül so viel Zeit wie $M_{\text{xor}}$, macht also weniger als quadratisch viele Schritte.

Das ist aber ein Widerspruch zur Voraussetzung. $L_{\text{xor}}$ kann also nicht in weniger als quadratischer Zeit entschieden werden.

## **Problem 3:** Grammatiken $3 + 1 + 2 + 5 + 3 = 14$ Punkte

Für eine Funktion $f\colon \mathbb{N}_0 \to \mathbb{N}_+$ bezeichnen wir eine Grammatik $G = (\Sigma, V, S, R)$ als $f(n)$-*Grammatik*, wenn es für jedes Wort $w \in L(G)$ eine Ableitung $S \overset{*}{\to} w$ in höchstens $f(|w|)$ Ableitungsschritten gibt. Außerdem bezeichnen wir eine Grammatik als $\mathcal{O}(f(n))$-*Grammatik*, wenn sie eine $g(n)$-Grammatik ist mit $g(n) \in \mathcal{O}(f(n))$.

(a) Zeigen Sie, dass die folgende Grammatik $G$ mit $L(G) = \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_0\}$ eine $\mathcal{O}(n^2)$-Grammatik, aber keine $\mathcal{O}(n)$-Grammatik ist.

$G = (\Sigma, V, S, R)$ mit $\Sigma = \{\mathsf{a}, \mathsf{b}\}, V = \{S, X, A, B\}$ und

$$
\begin{aligned}
R = \{ \quad S &\to \mathsf{a}BX \mid \varepsilon, \\
X &\to ABX \mid \varepsilon, \\
BA &\to AB, \\
\mathsf{a}A &\to \mathsf{aa}, \\
\mathsf{a}B &\to \mathsf{ab}, \\
\mathsf{b}B &\to \mathsf{bb} \\
\}
\end{aligned}
$$

(b) Geben Sie eine $\mathcal{O}(n)$-Grammatik für $L = \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_0\}$ an und zeigen Sie, dass Ihre Grammatik tatsächlich eine $\mathcal{O}(n)$-Grammatik ist.

(c) Zeigen Sie, dass es für jede kontextfreie Sprache $L$ eine $\mathcal{O}(n)$-Grammatik gibt, die $L$ erzeugt.

(d) Zeigen Sie, dass eine Sprache $L$ genau dann in $\mathsf{NP}$ ist, wenn es eine $p(n)$-Grammatik gibt, die $L$ erzeugt, wobei $p$ ein Polynom ist. Sie dürfen das folgende Theorem ohne Beweis verwenden.

**Theorem 1.** *Für jede nichtdeterministische Turingmaschine $\mathcal{M}$ gibt es eine Grammatik $G$ mit $L(G) = L(\mathcal{M})$. Dabei hat die Grammatik $G$ für jeden Berechnungspfad der Länge $\ell$ von $\mathcal{M}$, der ein Wort $w$ akzeptiert, eine Ableitung $S \overset{*}{\to} w$ der gleichen Länge $\ell$.*

(e) Sei $f$ eine berechenbare Funktion. Zeigen Sie: Wenn $G$ eine $f(n)$-Grammatik ist, dann gibt es eine nichtdeterministische Turingmaschine, bei der jeder Berechnungspfad terminiert und die $L(G)$ akzeptiert.

*Hinweis: Sie dürfen in dieser Teilaufgabe mehrere Bänder verwenden, sofern dies sinnvoll ist.*

**Lösung:**

(a) Sei $w \in L(G)$ ein Wort der Länge $2n$. Wir zeigen, dass die Länge jeder Ableitung von $S$ zu $w$ in $\Theta(n^2)$ liegt.

Zunächst beobachten wir, dass die ersten beiden Regeln die einzigen sind, die neue Variablen oder Terminale erzeugen, während es keine Regeln gibt, die das aktuell abgeleitete Wort kürzer machen. Außerdem ist die erste Regel nur im ersten Schritt anwendbar. Folglich gibt es nur eine Möglichkeit, $2n$ Variablen/Terminale zu erzeugen, nämlich einmal die erste Regel und $n$ Mal die zweite (inkl. Ableitung nach $\varepsilon$).

Zusätzlich sehen wir, dass die letzten drei Regeln die einzigen sind, die Variablen zu Terminalen ableiten (außer dem schon betrachteten ersten Schritt). Es gibt andersherum keine Möglichkeit, aus einem Terminal wieder eine Variable zu machen. Wir haben also in $S \overset{*}{\to} w$ genau $2n - 1$ solcher Ableitungen.

Wir bezeichnen nun für $i \geqslant 2, j \geqslant 1$ mit $A_i$ bzw. $B_j$ die Variable $A$ bzw. $B$, die im $i$-ten Ableitungsschritt von Regeln 1 oder 2 entstanden ist. Die letzten drei Regeln sind so konstruiert, dass sie ein $A$ nicht ableiten können, wenn es nach einem $B$ kommt. Also muss für jedes $j \geqslant 1$ die Variable $B_j$

mit allen $A_i$, $i > j$, mit Hilfe der dritten Regel getauscht werden. Da es keine Regel gibt, die eine Vertauschung von $A$ und $B$ mit $A$ vor $B$ erlaubt, wird jede der beschriebenen Vertauschungen genau einmal durchgeführt. Wir haben also in jeder Ableitung zu $w$ genau $\sum_{j=1}^{n-1} j - 1$ Anwendungen der dritten Regel.

Insgesamt ergibt das $1 + n + (2n - 1) + \sum_{j=1}^{n-1} j - 1 = n^2/2 + (5n)/2 - 1$ Ableitungsschritte.

(b) $G = (\Sigma, \{S\}, S, R)$ mit $R = \{S \to \mathsf{a}S\mathsf{b} \mid \varepsilon\}$

Ein Wort $\mathsf{a}^n\mathsf{b}^n \in L(G)$ kann durch $n$ Anwendungen der Regel $S \to \mathsf{a}S\mathsf{b}$ und eine Anwendung der Regel $S \to \varepsilon$ von $S$ abgeleitet werden, also in $n + 1 \in \mathcal{O}(n)$ Schritten.

(c) Für jede kontextfreie Sprache gibt es eine Grammatik in erweiterter Chomsky-Normalform. Diese hat Regeln der Form $A \to BC$ oder $A \to a$ mit $A, B, C \in V$ und $a \in \Sigma$ sowie ggf. $S \to \varepsilon$. Insbesondere gibt es keine Möglichkeit, Variablen oder Terminale zu löschen. Um ein Wort der Länge $n$ abzuleiten, werden also genau $n$ Variablen erzeugt, die zu genau $n$ Terminalen abgeleitet werden. Es wird also $n-1$ Mal eine Regel der Form $A \to BC$ und außerdem $n$ Mal eine Regel der Form $A \to a$ verwendet. Jede Ableitung (außer $S \to \varepsilon$) enthält also genau $2n - 1 \in \mathcal{O}(n)$ Schritte.

(d) *Anmerkung: Bis auf die Länge der Berechnungspfade/Ableitungen ist Theorem 1 bereits aus der Vorlesung bekannt.*

„$\Leftarrow$“ Sei $G$ eine Typ-poly Grammatik. Wir konstruieren eine nichtdeterministische Turingmaschine $\mathcal{M}$, die die Eingabe $w$ in polynomieller Zeit akzeptiert, falls $w \in L(G)$, und sonst nicht akzeptiert. Dazu schreibt $\mathcal{M}$ zunächst das Startsymbol $S$ auf das Band. In jedem Schritt wählt $\mathcal{M}$ nichtdeterministisch eine Regel aus und ebenfalls nichtdeterministisch eine Teilfolge des aktuellen Wortes auf dem Band, auf die die ausgewählte Regel anwendbar ist. Sobald auf dem Band $w$ steht, akzeptiert $\mathcal{M}$. Da $G$ eine Typ-poly Grammatik ist, gibt es für alle $w \in L(G)$ eine Ableitung polynomieller Länge, d.h. in $\mathcal{M}$ einen akzeptierenden Berechnungspfad polynomieller Länge. Außerdem akzeptiert $\mathcal{M}$ nach Konstruktion genau die Wörter, die von $S$ aus ableitbar sind. Also ist $\mathcal{M}$ eine nichtdeterministische Turingmaschine, die $L(G)$ in Polynomialzeit akzeptiert, das heißt $L(G) \in \mathsf{NP}$.

„$\Rightarrow$“ Sei nun $L \in \mathsf{NP}$. Wir konstruieren eine Typ-poly Grammatik, die $L$ erzeugt. Da $L$ in $\mathsf{NP}$ ist, gibt es eine nichtdeterministische Turingmaschine $\mathcal{M}$ und ein Polynom $p(n)$, sodass $\mathcal{M}$ für jedes Wort aus $L$ einen akzeptierenden Berechnungspfad der Länge $\leqslant p(n)$ hat und alle anderen Wörter nicht akzeptiert. Nach Theorem 1 gibt es eine Grammatik $G$ mit $L(G) = L(\mathcal{M})$, sodass es für jeden akzeptierenden Berechnungspfad von $\mathcal{M}$ eine Ableitung gleicher Länge in $G$ gibt. Insbesondere gibt es also für jedes Wort in $L$ eine Ableitung der Länge höchstens $p(n)$. Damit ist $G$ eine Typ-poly Grammatik, die $L$ erzeugt.

(e) Wir konstruieren eine Turingmaschine, die $L(G)$ akzeptiert. Sei $w \in \Sigma^*$ die Eingabe. Da $f$ berechenbar ist, gibt es eine Turingmaschine, die $f$ berechnet. Diese simulieren wir mit Eingabe $|w|$ und merken uns das Ergebnis auf einem separaten Band.

Nun schreiben wir $S$ auf das Band. In jedem Schritt wählt $\mathcal{M}$ nichtdeterministisch eine Regel aus und ebenfalls nichtdeterministisch eine Teilfolge des aktuellen Wortes auf dem Band, auf die die ausgewählte Regel anwendbar ist. Außerdem wird mit jedem Schritt die Zahl auf dem zweiten Band um eins verringert. Sobald auf dem Band $w$ steht, akzeptiert $\mathcal{M}$. Wenn auf dem zweiten Band eine 0 steht, auf dem ersten Band aber nicht $w$, so lehnen wir ab.

Falls $w \in L(G)$, so gibt es eine Ableitung in höchstens $f(|w|)$ Schritten, da $G$ eine Typ-$f(n)$ Grammatik ist. Also gibt es auch einen akzeptierenden Berechnungspfad, der $w$ auf das Band schreibt bevor der Zähler auf dem zweiten Band bei 0 ist, das heißt $w$ wird tatsächlich akzeptiert.

Falls $w \notin L(G)$, so gibt es keine Ableitung von $S$ zu $w$. Da wir aber nur akzeptieren, wenn wir eine Ableitungsfolge von $S$ zu $w$ gefunden haben, gibt es keinen akzeptierenden Berechnungspfad. Die konstruierte Turingmaschine akzeptiert $w$ also nicht.

## **Problem 4:** NP-Vollständigkeit $1 + 1 + 6 = 8$ Punkte

Wir betrachten das folgende Problem, von dem wir zeigen wollen, dass es NP-vollständig ist.

| ColumnFlipping | |
|---|---|
| **Gegeben:** | Eine $(2 \times n)$-Matrix mit Einträgen in $\mathbb{N}_0$, wobei $n \geqslant 1$ |
| **Frage:** | Für jede Spalte darf nun entschieden werden, ob die beiden Einträge vertauscht werden. Gibt es eine Menge $S$ von Spalten, sodass das Vertauschen der beiden Einträge für jede Spalte in $S$ dazu führt, dass die Summe beider Zeilen gleich ist? |

Beispiel:

Die Matrix $M$ ist eine Ja-Instanz, da die Wahl von $S$ als erste und dritte Spalte dazu führt, dass die Summe beider Zeilen jeweils 11 ist.

$$M = \begin{pmatrix} 2 & 1 & 0 & 0 & 3 & 4 \\ 0 & 0 & 3 & 9 & 0 & 0 \end{pmatrix} \quad \rightsquigarrow \quad \begin{pmatrix} 0 & 1 & 3 & 0 & 3 & 4 \\ 2 & 0 & 0 & 9 & 0 & 0 \end{pmatrix} \quad \begin{matrix} \Sigma = 11 \\ \Sigma = 11 \end{matrix}$$

(a) Konstruieren Sie eine Nein-Instanz von ColumnFlipping.

Sie wissen außerdem aus der Vorlesung, dass Partition NP-vollständig ist.

| Partition | |
|---|---|
| **Gegeben:** | Eine endliche Menge $M$,<br>eine Gewichtsfunktion $w\colon M \to \mathbb{N}_0$ |
| **Frage:** | Existieren disjunkte Teilmengen $M_1, M_2 \subseteq M$ mit $M_1 \cup M_2 = M$ und $w(M_1) = w(M_2)$? |
| Für eine Menge $X$ ist hierbei $w(X)$ definiert als $w(X) = \sum_{x \in X} w(x)$. | |

Wir zeigen nun schrittweise, dass ColumnFlipping ebenfalls NP-vollständig ist.

(b) Zeigen Sie, dass ColumnFlipping in NP liegt. Geben Sie dabei explizit an, was der Lösungsvorschlag des Orakels enthält.

(c) Geben Sie eine polynomielle Transformation von Partition zu ColumnFlipping an und zeigen Sie, dass Ihre Transformation tatsächlich eine korrekte, polynomielle Transformation ist.

*Hinweis: Betrachten Sie das Beispiel noch einmal.*

- Geben Sie hier Ihre polynomielle Transformation an (ohne Beweis[1]):
- Zeigen Sie nun, dass Ihre Transformation tatsächlich eine korrekte, polynomielle Transformation ist:

**Lösung:**

(a) $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$

(b) Der Lösungsvorschlag besteht aus einer Menge $S$ von Spalten, dargestellt als Nummer der Spalte. Wir vertauschen nun in Linearzeit die beiden Einträge der angegebenen Spalten und summieren anschließend die beiden Zeilen. Die Ausgabe ist ja, falls die Summen gleich sind, sonst nein.

[1] Falls Sie aus Versehen an dieser Stelle einen Beweis geschrieben haben, kennzeichnen Sie eindeutig, welcher Teil zur Transformation gehört und welcher Teil zum Beweis.

(c) **Transformation.** Gegeben ist eine PARTITION-Instanz $(M, w)$. Wir konstruieren eine COLUMNFLIPPING-Instanz $M'$. Seien dazu $m_1, \ldots, m_{|M|}$ die Elemente von $M$. Wir definieren $M'_{1,j} = w(m_j)$ und $M'_{2,j} = 0$ für $j = 1, \ldots, |M|$ Das heißt in der ersten Zeile von $M'$ stehen die Gewichte der PARTITION-Instanz, die zweite Zeile enthält nur Nullen.

*Hinweis: Eine Transformation muss immer alle möglichen Eingaben (hier: beliebige Instanzen von* PARTITION*) abbilden. Es genügt also nicht nur Ja-Instanzen abzubilden. Es ist nicht einmal möglich, in der gegebenen Zeit zu testen, ob es sich um eine Ja-Instanz handelt! Wenn in der Transformation also eine Partition $M_1 \dot\cup M_2 = M$ mit $w(M_1) = w(M_2)$ verwendet wurde, dann ist dies keine Transformation, da nur Ja-Instanzen solche $M_1, M_2$ haben und abgebildet werden. Zusätzlich sind $M_1, M_2$ nicht in polynomieller Zeit berechenbar (außer $\mathsf{P} = \mathsf{NP}$). Merkhilfe: Denken Sie an die Anwendung einer Transformation: Wir wollen zeigen, dass* PARTITION *in polynomieller Zeit lösbar wäre, wenn dies für* COLUMNFLIPPING *gelten würde. Gegeben ist also eine* beliebige PARTITION*-Instanz $I$ und wir wollen mit Hilfe der Transformation einen Algorithmus bauen, der entscheidet, ob $I$ eine Ja-Instanz ist. Es wäre also nicht sinnvoll, zu Beginn der Transformation zu testen, ob $M_1, M_2$ für $I$ existieren, d.h. ob $I$ eine Ja-Instanz ist. Wenn wir das könnten, dann bräuchten wir die Transformation nicht.*

**Beweis.** Zunächst beobachten wir, dass die Transformation polynomiell ist, da wir nur $2 \cdot |M|$ Einträge in jeweils konstanter Zeit setzen.

Sei nun $(M, w)$ eine PARTITION-Instanz und $M'$ die daraus konstruierte COLUMNFLIPPING-Instanz. Wir zeigen, dass $(M, w)$ genau dann eine Ja-Instanz ist, wenn $M'$ eine Ja-Instanz ist. Dabei bezeichnen wir die Elemente von $M$ wieder als $m_1, \ldots, m_{|M|}$.

„⇒“ Sei $M_1, M_2$ eine Lösung von $(M, w)$. Weiter seien $I_1, I_2 \subseteq \{1, \ldots, n\}$ die Mengen der Indizes der Elemente in $M_1$ bzw. $M_2$, das heißt $M_1 = \bigcup_{i \in I_1} m_i$ und $M_2 = \bigcup_{i \in I_2} m_i$. Wir zeigen, dass $S = I_2$ eine Lösung von $M'$ ist. Sei dazu $M^*$ die Matrix, die aus $M'$ entsteht, wenn die Einträge der Spalten in $S$ vertauscht werden. Dann gilt für $M^*$

$$\begin{aligned}\text{Summe erste Zeile} &= \sum_{j \in \{1,\ldots,n\} \setminus S} M^*_{1,j} = \sum_{j \in I_1} w(m_j) = \sum_{j \in I_2} w(m_j) = \sum_{j \in S} M^*_{2,j} \\ &= \text{Summe zweite Zeile},\end{aligned}$$

wobei das dritte Gleichheitszeichen ausnutzt, dass $M_1, M_2$ eine Lösung von $(M, w)$ ist, und das erste und letzte Gleichheitszeichen, dass alle anderen Einträge 0 sind.

„⇐“ Sei nun $S$ eine Lösung von $M'$ und $M^*$ die Matrix, die durch Vertauschen der Einträge in den entsprechenden Spalten entsteht. Wir zeigen, dass $M_2 = \{m_i \mid i \in S\}, M_1 = M \setminus M_2$ eine Lösung von $(M, w)$ ist. Wir haben $M_1, M_2$ so definiert, dass sie $M$ partitionieren, also bleibt zu zeigen, dass $w(M_1) = w(M_2)$ gilt.

$$w(M_2) = \sum_{j \in S} w(m_j) = \sum_{j=1}^{|M|} M^*_{2,j} = \sum_{j=1}^{|M|} M^*_{1,j} = \sum_{j \in \{1,\ldots,|M|\} \setminus S} w(m_j) = w(M_1),$$

wobei das dritte Gleichheitszeichen ausnutzt, dass $S$ eine Lösung von $M'$ ist.

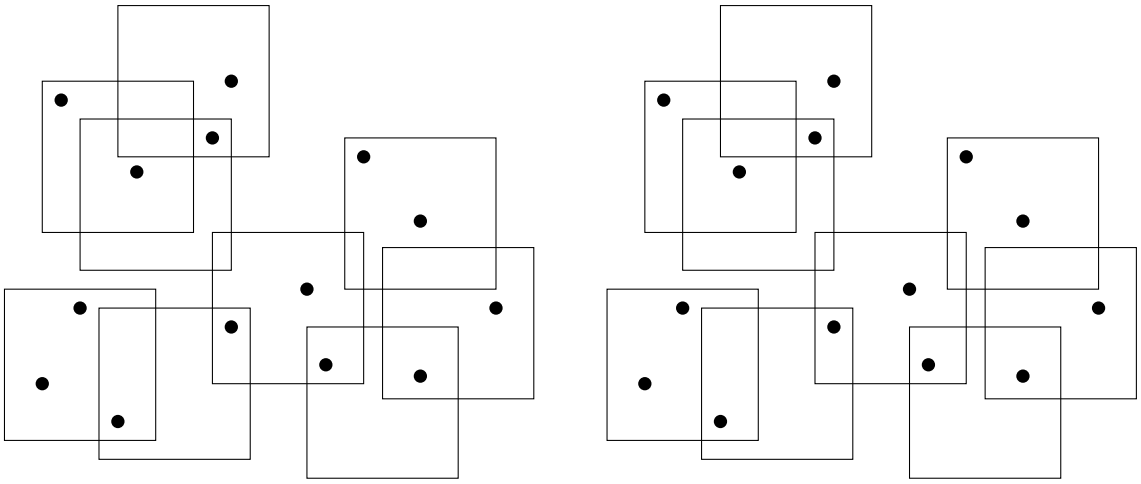
## **Problem 5:** Approximation $1 + 2 + 1 + 4 + 3 = 11$ Punkte

Bei dem Optimierungsproblem Unit Square Cover geht es darum, Punkte in der Ebene durch möglichst wenige Einheitsquadrate (mit Seitenlänge 1) zu überdecken.

| Unit Square Cover | |
|---|---|
| **Gegeben:** | • Menge von Punkten $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ in der Ebene, wobei $x_i, y_i \geqslant 0$ für alle $1 \leqslant i \leqslant n$<br>• Menge von achsenparallelen (Seiten sind parallel zur $x$- oder $y$-Achse) Einheitsquadraten $Q = \{q_1, \ldots, q_k\}$ mit Seitenlänge 1 |
| **Gesucht:** | Kardinalitätsminimale Teilmenge $Q^* \subseteq Q$, sodass jeder Punkt in $P$ von (mindestens) einem Quadrat aus $Q^*$ überdeckt wird<br>*Hinweis: Liegt ein Punkt auf dem Rand oder im Inneren eines gewählten Quadrats, wird dieser überdeckt.* |

(a) Markieren Sie für die Instanz in der folgenden Abbildung eine kardinalitätsminimale Teilmenge der gegebenen Einheitsquadrate, die die Punkte überdecken.

Falls Sie beide Kopien der Instanz bearbeiten, markieren Sie eindeutig die zu wertende Kopie.



Instanz für Teilaufgabe (a) (und eine zusätzliche Kopie)

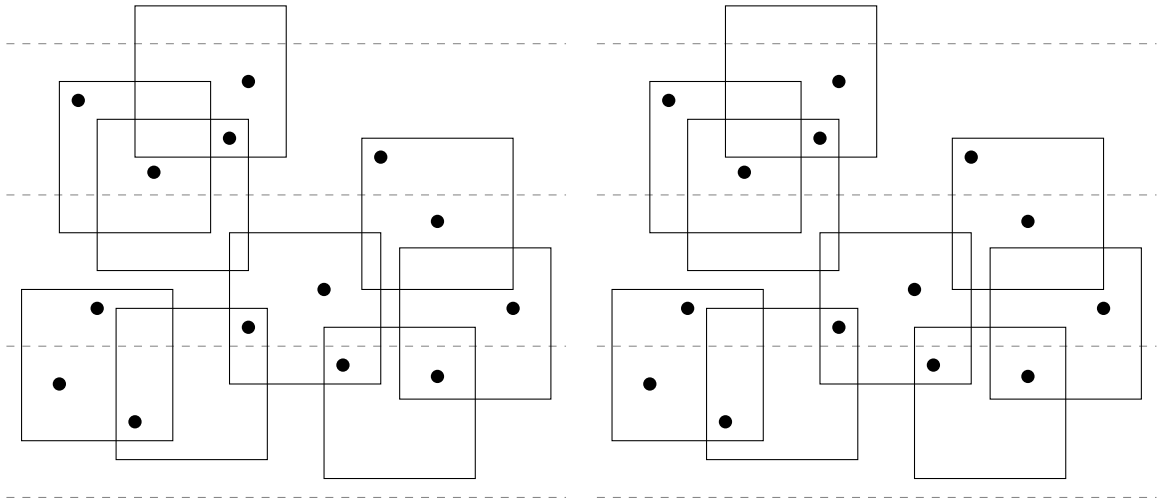
Betrachten Sie nun den polynomiellen Algorithmus $\mathcal{A}$.

**Algorithmus $\mathcal{A}$:** Unit Square Approx

```
1 Partitioniere das Koordinatensystem in Streifen S_0, S_1, ... der Höhe 1 mit
    S_i = {(x, y) | x ⩾ 0, y ∈ [i, i + 1)}
2 Für jeden Streifen S_i, der einen Punkt enthält
3   Bestimme eine optimale Lösung Q_i, die alle Punkte in S_i überdeckt
4 return Q_0 ∪ Q_1 ∪ Q_2 ∪ ... (Vereinigung aller Q_i)
```

(b) Markieren Sie für die Instanz in der folgenden Abbildung die Einheitsquadrate, die von $\mathcal{A}$ ausgewählt werden. Die Streifen sind schon vorgegeben und haben Höhe 1.

Falls Sie beide Kopien der Instanz bearbeiten, markieren Sie eindeutig die zu wertende Kopie.



Instanz für Teilaufgabe (b) (und eine zusätzliche Kopie)

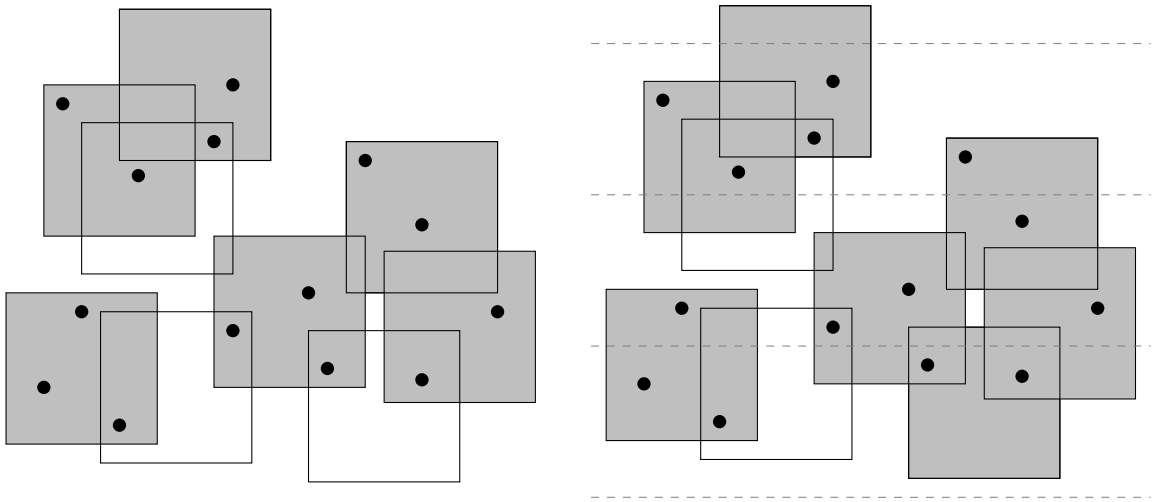
Sei für eine Instanz $I$ von Unit Square Cover $\mathrm{OPT}(I)$ eine optimale Lösung von $I$. Für jeden Streifen $S_i$ definieren wir die Menge $\mathrm{OPT}_i \subseteq \mathrm{OPT}(I)$ von Quadraten, die in der optimalen Lösung von $I$ einen Punkt in $S_i$ überdeckt.

(c) Zeigen Sie, dass $|Q_i| \leqslant |\,\mathrm{OPT}_i\,|$ gilt. Dabei bezeichnet $Q_i$ die Menge der Quadrate, die der Algorithmus $\mathcal{A}$ für Streifen $S_i$ wählt.

(d) Zeigen Sie, dass $\mathcal{A}$ ein Approximationsalgorithmus für Unit Square Cover mit relativer Güte 2 ist. Sie müssen nicht zeigen, dass $\mathcal{A}$ in polynomieller Zeit läuft.

(e) Sie dürfen annehmen, dass Unit Square Cover NP-vollständig ist. Zeigen Sie, dass es keinen polynomiellen Approximationsalgorithmus mit absoluter Güte $c \in \mathbb{N}$ gibt, falls $\mathsf{P} \neq \mathsf{NP}$.

**Lösung:**



Lösung für Teilaufgabe (a) Lösung für Teilaufgabe (b)

(c) Da $\mathrm{OPT}(I)$ alle Punkte von $I$ überdeckt und $\mathrm{OPT}_i$ die Einschränkung auf $S_i$ ist„ überdeckt $\mathrm{OPT}_i$ alle Punkte in $S_i$ und ist damit eine Lösung für die Punkte in $S_i$. Da $Q_i$ eine optimale Lösung für die Punkte in $S_i$ ist, ist jede andere Lösung mindestens so groß, insbesondere gilt $|Q_i| \leqslant |\,\mathrm{OPT}_i\,|$.

(d) Wir zeigen zunächst, dass $\sum_i |\operatorname{OPT}_i| \leqslant 2 \cdot |\operatorname{OPT}(I)|$ gilt. Jedes Quadrat in $\operatorname{OPT}_i$ kann höchstens Punkte in zwei Streifen überdecken, da die Seitenlänge der Quadrate gleich der Höhe der Streifen ist. Damit kann in der Summe $\sum_i |\operatorname{OPT}_i|$ jedes Quadrat höchstens zwei Mal gezählt werden. Es gilt damit $\sum_i |\operatorname{OPT}_i| \leqslant 2 \cdot |\operatorname{OPT}(I)|$.

Da in jedem Streifen alle Punkte durch gewählte Quadrate überdeckt werden, ist die Vereinigung aller $Q_i$ eine Lösung von $I$.

Für die Güte von $\mathcal{A}$ gilt:

$$|\mathcal{A}(I)| = \sum_i |Q_i| \leqslant \sum_i |\operatorname{OPT}_i| \leqslant 2 \cdot \operatorname{OPT}(I) \; .$$

Damit ist $\mathcal{A}$ ein polynomieller Approximationsalgorithmus mit relativer Güte 2.

(e) Wir nehmen an, es gibt einen Approximationsalgorithmus $\mathcal{A}'$ mit konstanter Güte $c \in \mathbb{N}$. Gegeben sei eine beliebige Instanz $I$. Wir kopieren die Instanz $c+1$ Mal entlang der $x$-Achse, sodass sich die Kopien nicht überschneiden. Die Instanz wird also nach rechts verschoben und immer wieder kopiert. Dadurch erhalten wir $I'$. Wir wenden nun $\mathcal{A}'$ auf $I'$ an und bekommen dadurch eine Lösung $Q^*$ der Größe $\lfloor |\mathcal{A}'(I')|/(c+1) \rfloor$ für eine Kopie.

Offensichtlich gilt $\operatorname{OPT}(I') = (c+1) \cdot \operatorname{OPT}(I)$. Dann gilt $|Q^*| = |\mathcal{A}'(I')|/(c+1) \leqslant (|\operatorname{OPT}(I')| + c)/(c+1) = |\operatorname{OPT}(I)| + c/(c+1)$. Da $c/(c+1) < 1$ gilt und sowohl $|\mathcal{A}'(I')|/(c+1)$ als auch $|\operatorname{OPT}(I)|$ ganzzahlig ist, gilt $|Q^*| = |\operatorname{OPT}(I)|$. Damit wäre $Q^*$ aber eine optimale Lösung. Da der modifizierte Algorithmus immer noch polynomielle Zeit benötigt, wäre das ein Widerspruch zu der Annahme $\mathsf{P} \neq \mathsf{NP}$.

## **Problem 6:** Entscheidbarkeit $1 + 5 + 3 = 9$ Punkte

Sei $\Sigma = \{0, 1\}$ ein Alphabet. Sei $k \in \mathbb{N}_0$ beliebig, aber fest. Betrachten Sie die folgende Sprache:

$$L_k = \{\langle \mathcal{M} \rangle \colon |L(\mathcal{M})| \geqslant k\}$$

(a) Welchen maximalen Chomsky-Typ hat die Sprache für $k = 0$? Begründen Sie kurz.

Aus der Vorlesung wissen Sie, dass die universelle Sprache

$$L_u = \{\langle \mathcal{M} \rangle \# w \colon w \in L(\mathcal{M})\}$$

nicht entscheidbar ist.

(b) Zeigen Sie, dass $L_k$ für $k > 0$ nicht entscheidbar ist, indem Sie von der universellen Sprache reduzieren.

(c) Zeigen Sie, dass $L_k$ semi-entscheidbar ist, indem Sie eine deterministische Turingmaschine konstruieren, die $L_k$ semi-entscheidet. Geben Sie dabei die Reihenfolge der Berechnungsschritte genau an. Sie müssen nicht beschreiben, wie Sie die Bänder verwalten.

**Lösung:**

(a) Für $k = 0$ ist $L_k = \Sigma^*$, da jede Sprache mindestens 0 Wörter enthält. Damit ist $L_0$ regulär, also vom Typ 3.

(b) Angenommen, $L_k$ wird von einer Turingmaschine $\mathcal{M}_k$ entschieden. Wir konstruieren eine Turingmaschine $\mathcal{M}_u$, die dann die universelle Sprache entscheidet. Für eine Eingabe $\langle \mathcal{M} \rangle \# w$, sei $\mathcal{M}_{\mathcal{M},w}$ die Turingmaschine, die jede Eingabe ignoriert, immer $\mathcal{M}$ auf $w$ simuliert und das Akzeptanzverhalten von $\mathcal{M}$ übernimmt. $\mathcal{M}_u$ simuliert dann $\mathcal{M}_k$ auf der Eingabe $\langle \mathcal{M}_{\mathcal{M},w} \rangle$. Falls $\mathcal{M}_k$ akzeptiert, akzeptiert auch $\mathcal{M}_u$.

Falls $\langle \mathcal{M} \rangle \# w \in L_u$, dann ist $w \in L(\mathcal{M})$ und $L(\mathcal{M}_{\mathcal{M},w}) = \Sigma^*$. Damit ist $|L(\mathcal{M}_{\mathcal{M},w})| = |\Sigma^*| \geqslant k$ und $\mathcal{M}_k$ akzeptiert.

Falls umgekehrt $\langle \mathcal{M} \rangle \# w \notin L_u$ gilt, ist $w \notin L(\mathcal{M})$ und $L(\mathcal{M}_{\mathcal{M},w}) = \emptyset$. Damit ist $|L(\mathcal{M}_{\mathcal{M},w})| = 0 < k$ und $\mathcal{M}_k$ akzeptiert.

Damit entscheidet $\mathcal{M}_u$ die universelle Sprache, was aber ein Widerspruch zur Unentscheidbarkeit der universellen Sprache ist. Also ist $L_k$ unentscheidbar.

(c) Wir konstruieren eine Turingmaschine $\mathcal{M}_k$, die eine Eingabe $\langle M \rangle$ genau dann akzeptiert, falls $\langle M \rangle \in L_k$. Die Wörter in $\Sigma^*$ werden in kanonischer Reihenfolge bearbeitet, um zu prüfen, ob $\mathcal{M}$ sie akzeptiert. Zuerst simuliert $\mathcal{M}_k$ ein Schritt der gegebenen Turingmaschine $\mathcal{M}$ auf dem ersten aufgezählten Wort. Danach wird ein weiteres Wort aufgezählt und auf den bis jetzt aufgezählten Wörtern ein weiterer Schritt simuliert, usw. Außerdem wird bei jedem akzeptierten Wort ein Zähler um eins erhöht. Wurden mindestens $k$ Wörter akzeptiert, akzeptiert auch $\mathcal{M}_k$. Da $\mathcal{M}_k$ alle Wörter in $L_k$ akzeptiert, ist $L_k$ semi-entscheidbar.