1. 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	1	2	3	2	–	8					–	
Aufg. 2	2	1	3	1	2	9						
Aufg. 3	1	2	3	4	2	12						
Aufg. 4	2	7	2	–	–	11				–	–	
Aufg. 5	2	5	3	–	–	10				–	–	
Aufg. 6	2	4	3	1	–	10					–	
Σ						60						

Problem 1: Grammatiken und Automaten $1 + 2 + 3 + 2 = 8$ Punkte

Wir betrachten Grammatiken, deren Regeln die Form $X \to abY$ oder $X \to \varepsilon$ haben, wobei X, Y Variablen und a, b Terminale sind. Eine solche Grammatik nennen wir 2-*rechtslinear*.

Beispiel: $G_{\text{Bsp}} = (\Sigma, V, S, R)$ mit $\Sigma = \{0, 1\}$, $V = \{S, X, Y\}$ und

$$R = \{ \quad \begin{array}{lcll} S & \to & 10X & \\ X & \to & 11X \mid 01Y & \\ Y & \to & \varepsilon & \} \end{array}$$

(a) Geben Sie eine Sprache an, für die es eine rechtslineare Grammatik gibt, aber keine 2-rechtslineare. Begründen Sie jeweils.

(b) Geben Sie einen endlichen Automaten an, der die Sprache $L(G_{\text{Bsp}})$ erkennt.

(c) Sei G eine beliebige 2-rechtslineare Grammatik. Beschreiben Sie einen endlichen Automaten, der $L(G)$ erkennt. Geben Sie insbesondere die Zustandsmenge, den Startzustand und akzeptierende Zustände an und beschreiben Sie die Zustandsübergänge.

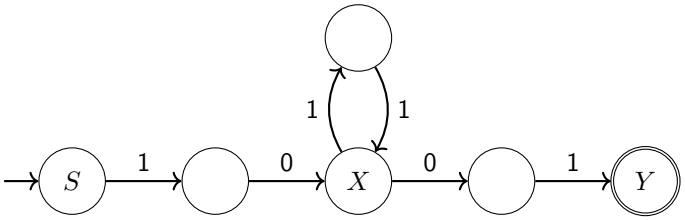
Nun erlauben wir Regeln der Form $X \to wY$ und $X \to \varepsilon$, wobei X, Y Variablen sind und $w \in \Sigma^*$ ein beliebiges Wort. Hierbei dürfen für unterschiedliche Regeln auch unterschiedliche Wörter gewählt werden. Solche Grammatiken nennen wir $*$-*rechtslinear*.

(d) Sind alle Sprachen, die von $*$-rechtslinearen Grammatiken erzeugt werden, regulär? Begründen Sie.

Lösung:

(a) $G = (\Sigma = \{\mathsf{a}\}, V = \{S, A\}, S, R = \{S \to \mathsf{a}A, A \to \varepsilon\})$ ist rechtslinear und erzeugt die Sprache $L(G) = \{\mathsf{a}\}$. Diese enthält ein Wort ungerader Länge. 2-Rechtslineare Grammatiken können aber nur Wörter gerader Länge erzeugen, also gibt es für $L(G)$ keine 2-rechtslineare Grammatik.

(b)



(c) Sei $G = (\Sigma, V, S, R)$ eine 2-rechtslineare Grammatik, wobei die $R_1, \ldots, R_r \in R$ die Regeln bezeichnen. Die Zustandsmenge Q besteht aus den Variablen von G, sowie einem weiteren Zustand q_i für jede Regel R_i, $i = 1, \ldots, r$, das heißt $Q = V \cup \{q_1, \ldots, q_r\}$. Als Startzustand wählen wir $q_0 = S$. Alle Variablen, die sich zu ε ableiten lassen, werden Endzustände, das heißt $F = \{X \in V \mid X \to \varepsilon \in R\}$. Für jede Regel R_i der Form $X \to abY$ mit $X, Y \in V$ und $a, b \in \Sigma$ fügen wir nun die Übergänge $\delta(X, a) = q_i$ und $\delta(q_i, b) = Y$ hinzu. Dabei können nichtdeterministische Übergänge entstehen.

(d) Wir zeigen, dass es für alle $*$-rechtslinearen Grammatiken eine äquivalente rechtslineare Grammatik gibt, das heißt die dazugehörige Sprache ist regulär. Sei $G = (\Sigma, V, S, R)$ eine $*$-rechtslineare Grammatik, $R_1, \ldots, R_r$ bezeichnen wieder die Regeln und w_i bezeichnet das Wort von Regel R_i, das heißt $R_i = X \to w_i Y$ für zwei Variablen X, Y. Für Regeln, die auf ε abbilden, ist $w_i = \varepsilon$. Wir definieren einen endlichen Automaten $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ mit $L(\mathcal{A}) = L(G)$ wie oben, mit dem Unterschied, dass die Pfade zwischen den Zuständen, die zu Variablen korrespondieren, länger sind. Auch wenn es keine allgemeine obere Schranke für die Länge der Pfade gibt, sind die Pfade für eine feste Grammatik höchstens so lang wie das längste Wort, das in den Regeln verwendet wird. Wir konstruieren also tatsächlich einen endlichen Automaten.

Wir setzen

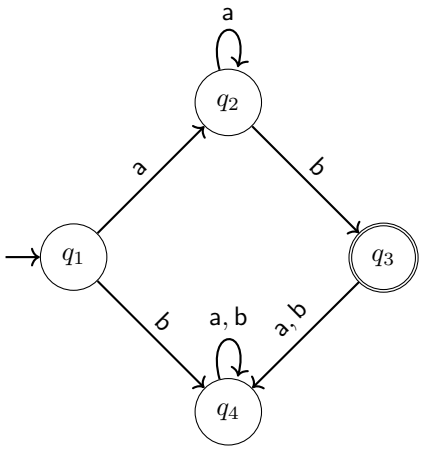
$$
\begin{aligned}
&Q = V \cup \{q_i^1, \ldots, q_i^{|w_i|} \mid i = 1, \ldots, r\},\\
&q_0 = S \text{ und}\\
&F = \{X \in V \mid X \to \varepsilon \in R\}.
\end{aligned}
$$

Für δ fügen wir für jede Regel R_i der Form $X \to w_i Y$ die Übergänge $X \to q_i^1 \to q_i^2 \to \cdots \to q_i^{|w_i|} \to Y$ ein, wobei das entsprechende Zeichen von w_i gelesen wird.

Problem 2: Sprachen von Nerodeklassen $2 + 1 + 3 + 1 + 2 = 9$ Punkte

In dieser Aufgabe geht es um die Nerode-Relation, die wir mit R_L bezeichnen. Nicht angegebene Übergänge bei endlichen Automaten führen in den (impliziten) Müllzustand, das heißt das Wort wird nicht akzeptiert.

(a) Im Folgenden ist ein minimaler Automat $\mathcal{A}$ gegeben, das heißt es gibt keinen Automaten mit weniger Zuständen, der $L(\mathcal{A})$ erkennt. Geben Sie die Äquivalenzklassen der Nerode-Relation $R_{L(\mathcal{A})}$ in Mengenschreibweise an.



(b) Äquivalenzklassen der Nerode-Relation sind Mengen von Wörtern, die wir als Sprachen auffassen können. Geben Sie einen DEA an, der die Sprache $L = [\mathsf{ba}]$ erkennt, wobei $[\mathsf{ba}]$ die Äquivalenzklasse von ba in $R_{L(\mathcal{A})}$ ist.

(c) Sei $\mathcal{B}$ ein DEA über dem Alphabet Σ und w ein Wort aus Σ^*. Ferner sei $[w]$ die Äquivalenzklasse von w bezüglich $R_{L(\mathcal{B})}$. Wir fassen diese nun als Sprache $L_w = [w]$ auf. Geben Sie ein Verfahren an, um einen Automaten zu konstruieren, der L_w erkennt.

(d) Wenn L eine reguläre Sprache ist, sind dann auch alle Äquivalenzklassen der Nerode-Relation R_L regulär? Begründen Sie.

(e) Zeigen Sie, dass Nerode-Äquivalenzklassen von kontextfreien Sprachen im Allgemeinen nicht regulär sind.

Lösung:

(a) $[\varepsilon] = \{\varepsilon\}$

$[\mathsf{a}] = \{\mathsf{a}^n \mid n \in \mathbb{N}_+\}$

$[\mathsf{ab}] = \{\mathsf{a}^n\mathsf{b} \mid n \in \mathbb{N}_+\} = L(\mathcal{B})$

$[b] = \Sigma^* \setminus ([\varepsilon] \cup [\mathsf{a}] \cup [\mathsf{ab}])$

(b) $\mathcal{A}$ mit akzeptierendem Zustand q_4 statt q_1 erkennt die Äquivalenzklasse von ba in $R_{L(\mathcal{A})}$.

(c) Konstruiere den minimalen Automaten der Nerode-Relation zu $\mathcal{B}$. Jeder Zustand in diesem DEA $\mathcal{B}'$ entspricht einer Nerode Äquivalenzklasse. Finde die Äquivalenzklasse von w, indem w als Eingabe für $\mathcal{B}'$ verwendet wird, der Zustand, nach Abarbeitung von w entspricht $[w]$. Setze diesen Zustand als einzigen akzeptierenden Zustand des Automaten und er erkennt L_w.

(d) Da nach (c) jede Äquivalenzklasse der Nerode-Relation einer regulären Sprache von einem DEA erkannt werden kann, sind diese Äquivalenzklassen selbst regulär.

(e) In der kontextfreien Sprache $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}_+\}$ ist die Nerode Äquivalenzklasse von $\mathtt{ab}$ die Sprache L selbst, die kontextfrei und nicht regulär ist. Somit sind Nerode Äquivalenzklassen von beliebigen Sprachen nicht zwangsläufig regulär.

Problem 3: Kompressions-Turingmaschine $1 + 2 + 3 + 4 + 2 = 12$ Punkte

Wir interessieren uns für fehlerfreie Kompression. Wir suchen also berechenbare Funktionen, die Eingabeworte in eine kompaktere Darstellung umwandeln. Es sollte möglich sein, diese Komprimierung rückgängig zu machen, also sollten unsere Kompressionsfunktionen injektiv sein. Daher nennen wir eine Funktion *Kompressionsfunktion*, wenn sie berechenbar und injektiv ist.

Betrachten wir nun die Kompressionsfunktion $c : \{\mathsf{a}, \mathsf{b}\}^* \to \{\mathsf{a}, \mathsf{b}, \mathsf{0}, \mathsf{1}\}^*$, die jeden inklusionsmaximalen Block der Länge $k \in \mathbb{N}^+$ aus aufeinanderfolgenden gleichen Zeichen $i \in \{\mathsf{a}, \mathsf{b}\}$ auf $i \cdot \mathrm{bin}(k)$ abbildet. Hierbei steht $\mathrm{bin}(n)$ für die Binärdarstellung von n.

Beispiel: $c(\mathsf{abbaaa}) = \mathsf{a1b10a11}$.

Die Notation $|w|$ bezeichnet wie üblich die Länge des Wortes w.

(a) Sei $L = \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_+\}$. Geben Sie für Wörter $w \in L$ die Länge von $c(w)$ in Abhängigkeit von $|w|$ im $\mathcal{O}$-Kalkül an.

(b) Geben Sie für jedes $i \in \mathbb{N}_+$ ein Wort $w_i \in \{\mathsf{a}, \mathsf{b}\}^*$ mit $|w_i| \geqslant i$ an, sodass $|c(w_i)| \geqslant |w_i|$. Begründen Sie.

Wir betrachten nun Turingmaschinen mit einer Kompressionsoperation, die eine Kompressionsfunktion realisiert. Eine Kompressionsoperation kann zu einem beliebigen Zeitpunkt aufgerufen werden und ersetzt die Eingabe w auf dem Band der Turingmaschine in einem Schritt durch das Ergebnis der Kompressionsfunktion. Nehmen Sie an, wir haben eine Kompressionsoperation C zur Verfügung, welche das Wort w auf dem Band durch $c(w)$ ersetzt, wobei c die oben definierte Kompressionsfunktion ist.

(c) Geben Sie an, wie eine deterministische Turingmaschine (mit einem Band) das Wortproblem von $L = \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_+\}$ möglichst effizient im $\mathcal{O}$-Kalkül lösen kann, wenn sie die Operation C benutzen darf.

Im nächsten Schritt konstruieren Sie selbst eine Turingmaschine, die eine Kompressionsfunktion berechnet, sowie eine Turingmaschine, die die Kompression rückgängig macht.

(d) Sei $\Sigma = \{\mathsf{a}, \mathsf{b}, \#\}, \Gamma = \Sigma \cup \{*\}$ und $L' = \{w\#w \mid w \in \{\mathsf{a}, \mathsf{b}\}^*\}$. Geben Sie eine Kompressionsfunktion $f \colon \Sigma^* \to \Gamma^*$ an, die alle Wörter aus L' auf maximal die Hälfte ihrer Länge komprimiert. Sie dürfen also für die Kompression zusätzlich zu den Zeichen in Σ das Zeichen $*$ verwenden. Beachten Sie, dass f für alle Wörter aus Σ^* definiert sein muss, aber die Anforderung an die Länge nur für Wörter aus L' gilt. Beschreiben Sie außerdem eine deterministische Turingmaschine $\mathcal{M}$, die f berechnet, sowie eine deterministische Turingmaschine $\overline{\mathcal{M}}$, die das Inverse von f berechnet.

Erinnerung: Kompressionsfunktionen müssen injektiv sein.
Hinweis: Sie dürfen in dieser Teilaufgabe mehrere Bänder verwenden.

Wir haben in den ersten beiden Teilaufgaben gesehen, dass die Kompressionsfunktion c besonders gut für Wörter der Sprache $\{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}_+\}$ ist, aber manche anderen Wörter nicht kürzt. Wir zeigen nun, dass es nicht für jede Sprache L eine Kompressionsfunktion gibt, die genau die Wörter aus L kürzt.

(e) Sei $\mathcal{H} \subseteq \{\mathsf{0}, \mathsf{1}, \#\}^*$ die Sprache des Halteproblems. Zeigen Sie, dass es keine Kompressionsfunktion gibt, die alle Wörter aus $\mathcal{H}$ auf ein kürzeres Wort abbildet, aber die Länge aller anderen Wörter unverändert lässt.

Lösung:

(a) Es gilt $c(\mathsf{a}^n\mathsf{b}^n) = \mathsf{a}\ \mathrm{bin}(n)\ \mathsf{b}\ \mathrm{bin}(n)$. Die Länge von $\mathrm{bin}(n)$ ist $1 + \lfloor \log_2(n) \rfloor$) also hat $c(w)$ für $w \in L$ Länge $2(2 + \lfloor \log_2(n) \rfloor) \in \mathcal{O}(\log(n)) = \mathcal{O}(\log |w|)$.

(b) Wörter, in denen a und b alternieren, werden durch c auf Wörter linearer Länge abgebildet. Formal sind das Wörter gerader Länge in $\{(\mathsf{ab})^{\frac{i}{2}} \mid i \in \mathbb{N}_+, i \equiv_2 0\}$ und Wörter ungerader Länge in $\{(\mathsf{ab})^{\frac{i-1}{2}}\mathsf{a} \mid i \in \mathbb{N}_+, i \equiv_2 1\}$. Jedes Zeichen a oder b in diesen Wörtern wird von c auf a1 beziehungsweise b1 abgebildet, also ist das Ergebnis der Kompression doppelt so lang, wie das Eingabewort.

(c) Auf der komprimierten Eingabe funktioniert die Erkennung in $\mathcal{O}(\log^2(n))$. Zuerst wird überprüft, ob die Eingabe die Form $\mathsf{a}\{0,1\}^+\mathsf{b}\{0,1\}^+$ hat (wenn nicht kann abgelehnt werden), dann werden die beiden Binärzahlen Zeichen für Zeichen verglichen, wenn ein Zeichenpaar nicht gleich ist, oder eine Binärzahl länger als die andere ist, lehnen wir ab. Diese Vergleiche benötigen pro Zeichenpaar $\mathcal{O}(\log(n))$ Kopfbewegungen.

(d) Sei $\Gamma = \Sigma \cup \{*\}$, dann können wir die Kompressionsfunktion f wie folgt definieren:

$$f(w) = \begin{cases} w' & \text{, wenn } \exists w' \in \Sigma^* \mid w = w'\#w' \\ w* & \text{, sonst} \end{cases}$$

Falls nicht genau ein # in der Eingabe ist, wird die Eingabe um $*$ am Ende ergänzt und sonst unverändert ausgegeben. Nehmen wir also an, dass es genau ein # gibt. $\mathcal{M}$ verwendet zwei Bänder. Auf dem ersten steht die Eingabe, die als erstes auf das zweite Band kopiert wird. Der Kopf auf dem zweiten Band geht auf das Feld rechts von #, der auf dem ersten Band auf das erste Feld. Nun vergleichen die beiden Köpfe das gelesene Zeichen und laufen einen Schritt nach rechts und wiederholen dies bis sie Blank oder # lesen. Falls die gelesenen Felder nicht übereinstimmen oder sie nicht gleichzeitig enden, so wird die Eingabe auf dem ersten Band um $*$ ergänzt und ausgegeben. Andernfalls wird alles nach # und # selbst auf dem ersten Band gelöscht und das Ergebnis auf dem ersten Band ausgegeben.

$\overline{\mathcal{M}}$ überprüft zuerst das letzte Zeichen der Eingabe. Wenn das Zeichen $*$ ist, entfernen wir dieses $*$ und sind fertig. Wenn das letzte Symbol nicht $*$ ist, schreiben wir # hinter die Eingabe und kopieren das Wort davor auf die Felder nach #. Dazu wird schrittweise das erste nicht markierte Symbol in $\{\mathsf{a}, \mathsf{b}\}$ von der Eingabe markiert und das gleiche Symbol ans Ende des Bandes kopiert. Wenn das Trennzeichen erreicht ist, werden alle Markierungen entfernt.

(e) Nehmen wir an, $c_{\mathcal{H}}$ ist eine Kompressionsfunktion, die alle Wörter in $\mathcal{H}$ auf kürzere Wörter abbildet, Wörter, die nicht in $\mathcal{H}$ liegen aber nicht kürzt. Mit $c_{\mathcal{H}}$ könnten wir das Halteproblem lösen, indem wir eine Eingabe w auf ein separates Band kopieren, sie zu $c_{\mathcal{H}}(w)$ umwandeln und vergleichen, ob $|w| < |c_{\mathcal{H}}(w)|$. Wenn ja, ist w in $\mathcal{H}$, sonst nicht. $c_{\mathcal{H}}$ ist berechenbar, also kann eine TM diese Berechnung durchführen, aber das Halteproblem ist nicht entscheidbar. Also haben wir einen Widerspruch zur Behauptung, dass $c_{\mathcal{H}}$ existiert.

Problem 4: NP-Vollständigkeit $2 + 7 + 2 = 11$ Punkte

Wir betrachten in dieser Aufgabe die folgenden zwei Probleme:

2-Fairbung

Gegeben: Ein Graph G

Frage: Gibt es eine 2-Färbung der Knoten, sodass genau die Hälfte der Knoten rot gefärbt ist und die andere Hälfte blau?

Eine 2-Färbung weist jedem Knoten eine Farbe aus $\{\text{rot}, \text{blau}\}$ zu, sodass je zwei benachbarte Knoten unterschiedliche Farben haben.

TupleSplitting

Gegeben: Eine Multi-Menge[1] M von Tupeln $(x, y) \in \mathbb{N}_+^2 \cup \{(0, 1), (1, 0)\}$

Frage: Können die Einträge der Tupel so in zwei Multi-Mengen R, B aufgeteilt werden, dass

- für jedes Tupel aus M genau einer der beiden Einträge in R ist und der andere in B (insbesondere gilt $|R| = |B| = |M|$) und
- die Summe der Elemente in R und B gleich ist, also $\sum_{r \in R} r = \sum_{b \in B} b$

Das heißt für jedes Tupel (x, y) gibt es zwei Optionen, nämlich $x \in R, y \in B$ oder $x \in B, y \in R$.

Beispiel:

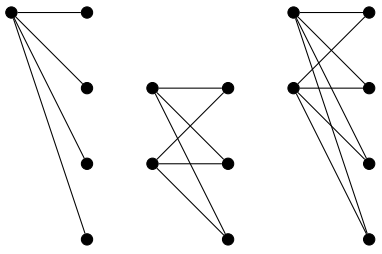
Die TupleSplitting-Instanz $M = \{(1, 4), (2, 3), (2, 4)\}$ hat die Lösung $R = \{1, 3, 4\}, B = \{4, 2, 2\}$.

Im folgenden Kasten sehen Sie eine Transformation von TupleSplitting zu 2-Fairbung.

Gegeben ist eine TupleSplitting-Instanz M. Wir konstruieren daraus eine 2-Fairbung-Instanz G wie folgt. G enthält für jedes Tupel $(x, y) \in M$ eine Kopie des vollständig bipartiten Graphen $K_{x,y}$. Der Graph G besteht also aus $|M|$ Zusammenhangskomponenten, die jeweils zu einem Tupel korrespondieren.

Hierbei bezeichnet $K_{x,y}$ einen Graphen, der wie folgt konstruiert wird: Wir nehmen zwei (disjunkte) Knotenmengen X und Y der Größe $|X| = x$ bzw. $|Y| = y$ und fügen alle möglichen Kanten zwischen den beiden Knotenmengen ein. Der Graph $K_{x,y}$ hat also eine eindeutige 2-Färbung (bis auf Vertauschung der Farben): x Knoten werden mit der ersten Farbe gefärbt, die restlichen y Knoten mit der zweiten Farbe.

Beispiel: $M = \{(1, 4), (2, 3), (2, 4)\}$ wird abgebildet auf den folgenden Graphen mit drei Zusammenhangskomponenten:



(a) Erklären Sie, warum die im Kasten angegebene Transformation nicht polynomiell ist.

[1] In einer Multi-Menge dürfen Elemente mehrfach vorkommen.

(b) UnaryTupleSplitting bezeichnet nun das Problem TupleSplitting, bei dem die Instanzen unär kodiert werden. Geben Sie eine polynomielle Transformation von 2-Fairbung zu UnaryTupleSplitting an und zeigen Sie, dass Ihre Transformation tatsächlich eine korrekte, polynomielle Transformation ist. Als Hilfestellung geben wir den ersten Teil der Transformation schon vor. Sie dürfen davon ausgehen, dass der vorgegebene Teil der Transformation in Polynomialzeit berechenbar ist.

- Vervollständigen Sie hier die polynomielle Transformation (ohne Beweis[2]):

  Gegeben ist eine 2-Fairbung-Instanz G. Wir konstruieren daraus eine UnaryTupleSplitting-Instanz M. Für jede Zusammenhangskomponente C des Graphen G konstruieren wir ein Tupel $(x, y) \in \mathbb{N}_+^2 \cup \{(0, 1), (1, 0)\}$. Wir prüfen zuerst, ob C 2-färbbar ist. Falls nein, so brechen wir ab und setzen $M = \{(0, 1)\}$. Andernfalls hat C eine eindeutige 2-Färbung Φ in rot und blau, bis auf Vertauschung der Farben.

- Zeigen Sie nun, dass Ihre Transformation tatsächlich eine korrekte, polynomielle Transformation ist:

(c) Gehen Sie davon aus, dass $\mathsf{P} \neq \mathsf{NP}$ gilt. Sie dürfen außerdem verwenden, dass TupleSplitting NP-vollständig ist, es aber einen pseudopolynomiellen Algorithmus gibt, der TupleSplitting löst. Ist 2-Fairbung ebenfalls NP-vollständig? Beweisen Sie.

Lösung:

(a) Die Transformation ist nicht polynomiell, da Zahlen in der TupleSplitting-Instanz logarithmische Länge haben (logarithmisch in der Größe der Zahl). Andersrum können die Zahlen also exponentiell groß sein in der Länge der Eingabe. Die Größe des konstruierten Graphen ist aber linear in der Größe der Zahlen, also ebenfalls exponentiell in der Länge der Kodierung der TupleSplitting-Instanz.

Anmerkung: Die Transformation ist tatsächlich lösungsäquivalent:

Sei M eine Ja-Instanz von TupleSplitting *und G der daraus konstruierte Graph. Für jedes Tupel $(x, y) \in M$ gibt es also eine Kopie $K_{x,y}$ in G mit Knotenpartition X, Y wie oben. Seien nun R, B eine Lösung von M. Wir konstruieren daraus eine Lösung für G. Dazu entscheiden wir für jede Zusammenhangskomponente $K_{x,y}$, die zu einem Tupel (x, y) gehört, welche der beiden Knotenmengen X, Y blau und welche rot sein soll. Falls $x \in R, y \in B$, dann färben wir alle Knoten in X rot und alle in Y blau. Andernfalls färben wir X blau und Y rot. Wir färben insgesamt also $\sum_{r \in R} r$ Knoten rot und $\sum_{b \in B} b$ Knoten blau. Da die Summen von R und B gleich sind, sind die Hälfte der Knoten in R.*

Sei nun $G = (V, E)$ eine Ja-Instanz, $\Phi \colon V \to \{blau, rot\}$ eine 2-Fairbung und M die TupleSplitting *Instanz, aus der G konstruiert wurde. Wir konstruieren eine Lösung R, B für M. Jede Zusammenhangskomponente von G ist eine Kopie von einem $K_{x,y}$, wobei (x, y) ein Tupel aus M ist. Wir beobachten, dass $K_{x,y}$ nur zwei mögliche Färbungen hat (die gleich sind bis auf Umbenennung der Farben). Falls die Knotenmenge der Größe x rot ist, so fügen für x zu R hinzu und y zu B, andernfalls andersrum. Auf diese Art fügen wir für jedes Tupel genau ein Element in jede Menge hinzu. Außerdem gilt $\sum_{r \in R} r =$ Anzahl roter Knoten $=$ Anzahl blauer Knoten $= \sum_{b \in B} b$, das heißt R, B ist eine Lösung von M.*

(b) **Transformation.** Gegeben ist eine 2-Fairbung-Instanz G. Wir konstruieren daraus eine UnaryTupleSplitting-Instanz M. Für jede Zusammenhangskomponente C von G konstruieren wir ein Tupel $(x, y) \in \mathbb{N}_+^2 \cup \{(0, 1), (1, 0)\}$. Wir prüfen zuerst, ob C 2-färbbar ist. Falls nein, so brechen wir ab und setzen $M = \{(0, 1)\}$. Andernfalls hat C eine eindeutige 2-Färbung Φ in rot und blau, bis auf

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

Vertauschung der Farben. Seien r und b die Anzahl roter bzw. blauer Knoten in Φ. Wir setzen nun $(x, y) = (r, b)$. Wir definieren nun M als die Menge aller so konstruierten Tupel.

Beweis. Sei G eine 2-FAIRBUNG-Instanz und M die daraus konstruierte UNARYTUPLESPLITTING-Instanz. Zunächst beobachten wir, dass M tatsächlich eine UNARYTUPLESPLITTING-Instanz ist: Falls eine Zusammenhangskomponente C mindestens zwei Knoten enthält, so enthält ihre 2-Färbung beide Farben und C wird auf ein Tupel in $\mathbb{N}_+^2$ abgebildet. Ein isolierter Knoten wird auf $(0, 1)$ oder $(1, 0)$ abgebildet.

Als nächstes zeigen wir, dass die Transformation polynomiell ist. Die Anzahl und Größe der Zusammenhangskomponenten in G ist höchstens linear. Zusammenhangskomponenten können – z.B. mit Breitensuche von noch nicht erreichten Knoten – in Linearzeit gefunden werden. Für jede Zusammenhangskomponente prüfen wir in Linearzeit mittels Breitensuche, ob es eine 2-Färbung gibt, berechnen diese ggf. und zählen die Größe der beiden Farbklassen. Die Länge der Unärkodierung der Tupel ist linear in der Anzahl der Knoten von G, kann also ebenfalls in Linearzeit konstruiert werden. Insgesamt haben wir für $O(n)$ Zusammenhangskomponenten jeweils $O(n)$ Aufwand, also $O(n^2)$.

Korrekturhinweis: Nur der letzte Teil mit der Unärkodierung ist gefordert, der Rest ist schon in der Aufgabenstellung gegeben.

Anmerkung: Tatsächlich benötigt die beschriebene Transformation nur Linearzeit.

Wir zeigen nun, dass G genau dann eine Ja-Instanz ist, wenn M ebenfalls eine Ja-Instanz ist.

„$\Rightarrow$“ Sei Φ eine 2-Fairbung von G. Wir konstruieren daraus eine Lösung R, B von M. Dabei entspricht R den roten Knoten und B den blauen Knoten. Sei also C eine Zusammenhangskomponente von G mit r roten und b blauen Knoten und (x, y) das daraus konstruierte Tupel. Da die 2-Färbung von C eindeutig ist, gilt $(x, y) = (r, b)$ oder $(x, y) = (b, r)$. Wir fügen nun r zu R hinzu und b zu B. Auf diese Art fügen wir für jedes Tupel genau ein Element in jede Menge hinzu. Außerdem gilt $\sum_{r \in R} r$ = Anzahl rote Knoten = Anzahl blaue Knoten = $\sum_{b \in B} b$, das heißt R, B ist eine Lösung von M.

„$\Leftarrow$“ Sei nun R, B eine Lösung von M. Insbesondere ist damit $M \neq \{(0, 1)\}$, was eine Nein-Instanz ist, also ist G 2-färbbar. Wir konstruieren eine 2-Fairbung Φ von G. Sei dazu C eine Zusammenhangskomponente von G und $(x, y) \in M$ das daraus konstruierte Tupel. Die beiden Farbklassen der (eindeutigen) 2-Färbung von C haben also die Größe x und y. Falls $x \in R$ und $y \in B$, so färben wir die Farbklasse mit x Knoten rot und die mit y Knoten blau, andernfalls tauschen wir rot und blau. Auf diese Weise erhalten wir eine zulässige 2-Färbung von C. Anschließend löschen wir x und y aus R und B. Wir färben also insgesamt $\sum_{m \in R} m$ Knoten rot und $\sum_{m \in B} m$ Knoten blau. Da R, B eine Lösung von M ist, haben wir gleich viele rote und blaue Knoten.

(c) 2-FAIRBUNG liegt in P: Wir transformieren eine gegebene Instanz I von 2-FAIRBUNG in polynomieller Zeit in eine unär kodierte TUPLESPLITTING-Instanz I'. Sei $\mathcal{A}$ ein pseudopolynomieller Algorithmus für TUPLESPLITTING. $\mathcal{A}$ löst also die unär kodierte Instanz I' in polynomieller Laufzeit. Da die Transformation lösungsäquivalent ist, ist die Ausgabe von $\mathcal{A}$ auch die korrekte Lösung von I.

Unter der Annahme $\mathsf{P} \neq \mathsf{NP}$ ist 2-FAIRBUNG also nicht NP-vollständig.

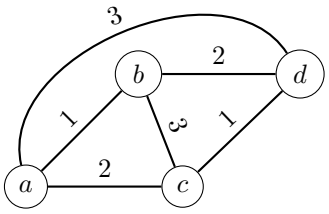
Problem 5: Approximation $2 + 5 + 3 = 10$ Punkte

Das Problem Edge Coloring sucht für einen gegebenen Graphen eine Färbung der Kanten mit möglichst wenigen Farben, sodass keine zwei Kanten mit gemeinsamem Endpunkt die gleiche Farbe haben.

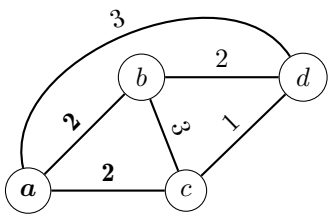
Edge Coloring	
Gegeben:	Ein endlicher Graph $G = (V, E)$
Gesucht:	Minimale Zahl an Farben, sodass alle Kanten gefärbt werden können und keine zwei Kanten mit gemeinsamem Endpunkt die gleiche Farbe haben.

Wir verwenden Zahlen in $\mathbb{N}_+$ als Farben.

Das Beispiel verdeutlicht die Definition, links ist eine gültige Färbung gegeben, rechts nicht.



Lösung von Edge Coloring mit 3 Farben



Keine Lösung von Edge Coloring (Knoten a hat zwei gleichfarbige Kanten)

Der Algorithmus Greedy Edge Coloring kann benutzt werden, um eine Kantenfärbung zu finden.

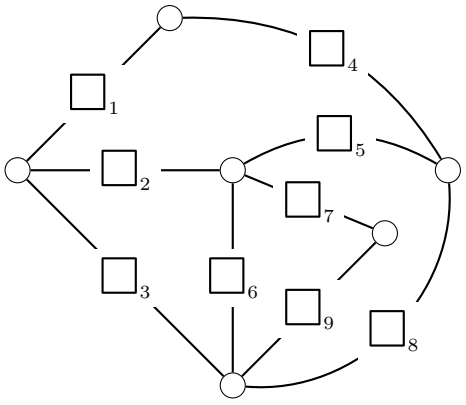
Wir sagen, eine Farbe i an einem Knoten v ist frei, wenn keine Kante an v schon mit Farbe i gefärbt ist.

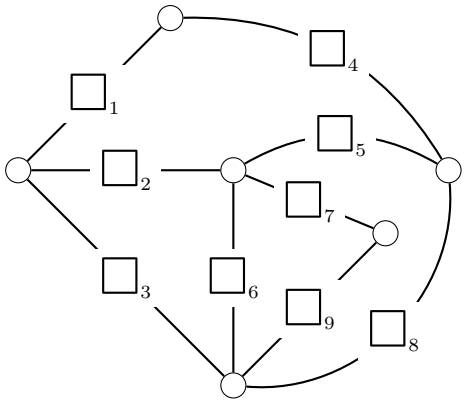
Algorithmus 1: Greedy Edge Coloring

```
1 Zu Beginn sind alle Kanten ungefärbt
2 solange es eine ungefärbte Kante gibt tue
3     Wähle beliebige ungefärbte Kante e = {v, w}
4     Weise e die kleinste Farbe zu, die bei v und w frei ist
5 return größte verwendete Farbe
```

(a) Führen Sie den Algorithmus Greedy Edge Coloring auf dem folgenden Graphen aus. Tragen Sie dafür die Farbe in das Kästchen ein. Die Zahlen neben den Kästchen geben die Reihenfolge an, in der Sie die Kanten betrachten sollen.

Hinweis: Falls Sie beide Kopien verwenden, kennzeichnen Sie eindeutig, welche Lösung zu werten ist.





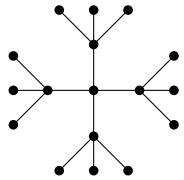
(b) Zeigen Sie, dass Greedy Edge Coloring ein polynomieller Approximationsalgorithmus mit relativer Gütegarantie von 2 für Edge Coloring ist.

Hinweis: Geben Sie eine untere Schranke an die optimale Färbung $\mathsf{Opt}(G)$ *in Abhängigkeit des größten Knotengrads* d_{max} *an.*

Hinweis: Zeigen Sie auch, dass Greedy Edge Coloring *eine korrekte Lösung ausgibt.*

(c) Zeigen Sie, dass Greedy Edge Coloring keine bessere relative Approximationsgüte als 2 hat.

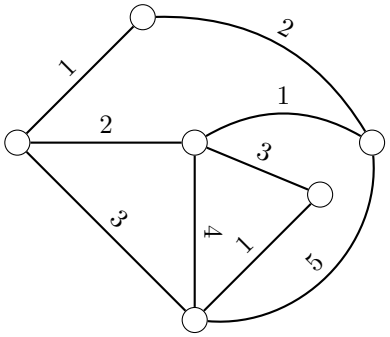
Hinweis: Betrachten Sie für jedes $\ell \in \mathbb{N}_+$ *einen Baum* T_ℓ*, dessen Wurzel* $\ell + 1$ *Kinder hat, die jeweils zu* ℓ *Blättern verbunden sind:*



Visualisierung von T_3

Lösung:

(a)



(b) Greedy Edge Coloring weist in jedem Schritt einer ungefärbten Kante eine Farbe zu, die noch keine adjazente Kante hat. Also wurde nach $|E|$ Schritten jeder Kante eine Farbe zugewiesen und keine zwei zum selben Knoten inzidenten Kanten haben die gleiche Farbe. Da jeder Schritt in $\mathcal{O}(d_{\max})$ läuft, ist die Laufzeit von Greedy Edge Coloring polynomiell.

Eine untere Schranke für die Anzahl der benötigten Farben ist der höchste Knotengrad $d_{\max}$ in G. Wir zeigen nun, dass Greedy Edge Coloring nie mehr als $2d_{\max}$ Farben verwendet. Eine Kante e hat maximal $2d_{\max} - 2$ inzidente Kanten und Greedy Edge Coloring berücksichtigt nur die Farben von inzidenten Kanten, also wird e maximal die Farbe $2d_{\max} - 1$ zugewiesen. Somit ist die relative Gütegarantie der Approximation von Greedy Edge Coloring maximal $\frac{2d_{\max}-1}{d_{\max}} < 2$.

(c) Für ein gegebenes ℓ sei T_ℓ der gewurzelte Baum mit $(\ell + 1)^2 + 1$ Knoten, Tiefe zwei und Grad $\ell + 1$ an allen inneren Knoten. Wenn Greedy Edge Coloring zuerst die Kanten zu den Blättern und danach die Kanten zur Wurzel abarbeitet, haben alle zur Wurzel inzidenten Kanten Farbe mindestens $\ell + 1$, also werden $2\ell + 1$ Farben verwendet. Eine optimale Lösung benötigt nur $\ell + 1$ Farben (färbe Kanten inzident zur Wurzel mit Farben 1 bis $\ell + 1$, dann hat jeder zur Wurzel adjazente Knoten ℓ ungefärbte inzidente Kanten zu Blättern, die mit den für diesen Knoten verbleibenden ℓ Farben gefärbt werden können). Die Güte ist also bestenfalls $(2\ell + 1)/(\ell + 1) = 2 - 1/(\ell + 1)$, was für große ℓ gegen 2 geht. Es gibt also für jedes $\varepsilon > 0$ ein ℓ, sodass die Güte schlechter als $2 - \varepsilon$ ist. Zusammen mit (b) ergibt sich eine relative Approximationsgüte von 2.

Problem 6: Entscheidbarkeit $2 + 4 + 3 + 1 = 10$ Punkte

Sie $\Sigma = \{0, 1\}$ ein Alphabet. Betrachten Sie die folgende Sprache

$$L_{\text{even}} = \{\langle \mathcal{M} \rangle \# w \mid \mathcal{M} \text{ akzeptiert } w \text{ nach einer geraden Anzahl von Kopfbewegungen}\}$$

Bei einer Kopfbewegung bewegt sich der Kopf nach links oder nach rechts. Falls der Kopf stehen bleibt, ist das keine Kopfbewegung. Beachten Sie, dass alle Turingmaschinen in dieser Aufgabe deterministisch sind, insbesondere beschreibt $\langle \mathcal{M} \rangle$ immer eine deterministische Turingmaschine.

(a) Zeigen Sie, dass L_{even} semi-entscheidbar ist. Geben Sie dazu eine deterministische Turingmaschine an, die L_{even} semi-entscheidet.

Hinweis: Sie dürfen mehrere Bänder verwenden.

Aus der Vorlesung wissen Sie, dass die universelle Sprache

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

unentscheidbar ist. Sie sollen im Folgenden zeigen, dass auch L_{even} unentscheidbar ist.

(b) Sei $\mathcal{M}$ eine gegebene deterministische Turingmaschine. Konstruieren Sie eine deterministische Turingmaschine $\mathcal{N}_\mathcal{M}$ *mit nur einem Band* und $L(\mathcal{N}_\mathcal{M}) = L(\mathcal{M})$, sodass für alle Wörter $w \in \Sigma^*$ gilt:

$$\mathcal{N}_\mathcal{M} \text{ akzeptiert } w \text{ nach einer geraden Anzahl von Kopfbewegungen} \iff \mathcal{M} \text{ akzeptiert } w.$$

Erklären Sie, warum Ihre Turingmaschine im Akzeptanzfall gerade viele Kopfbewegungen macht. Ihre Konstruktion muss aus $\mathcal{M}$ berechenbar sein.

(c) Zeigen Sie, dass L_{even} unentscheidbar ist, indem Sie von der universellen Sprache reduzieren.

(d) Ist das Komplement von L_{even} semi-entscheidbar? Beweisen Sie.

Lösung:

(a) Wir konstruieren eine Turingmaschine $\mathcal{N}$, die L_{even} semi-entscheidet. Sei $\langle \mathcal{M} \rangle \# w$ eine Eingabe. Die Turingmaschine besteht aus zwei Bändern, wobei auf dem ersten Band $\mathcal{M}$ auf w simuliert wird und auf dem zweiten Band die Parität der Anzahl der Kopfbewegungen gespeichert wird. Auf dem zweiten Band steht am Anfang der Berechnung eine 0. Wenn der Kopf von $\mathcal{M}$ sich bewegt, wird das Bit auf dem zweiten Band invertiert. Wenn $\mathcal{M}$ nicht akzeptiert, akzeptiert auch $\mathcal{N}$ nicht. Ansonsten akzeptiert $\mathcal{N}$ genau dann, wenn nachdem $\mathcal{M}$ akzeptiert, auf dem zweiten Band eine 0 steht, also wenn $\mathcal{M}$ gerade viele Kopfbewegungen gemacht hat.

Eine andere Möglichkeit ist eine Turingmaschine, die wie in Teilaufgabe (b) die Parität im Zustand speichert. Die Konstruktion ist bis auf den letzten Schritt exakt gleich. Die Zustände $(f, 1) \in F$ werden aber anders als in Teilaufgabe (b) nicht gesondert behandelt. Diese Turingmaschine akzeptiert, wenn die gegebene Turingmaschine $\mathcal{M}$ das gegebene Wort w akzeptiert und $\mathcal{M}$ dabei gerade viele Kopfbewegungen macht.

(b) Wir konstruieren die Turingmaschine $\mathcal{N}_\mathcal{M}$ für eine gegebene Turingmaschine $\mathcal{M}$, indem wir $\mathcal{M}$ modifizieren. Sei Q die Zustandsmenge von $\mathcal{M}$. Für jeden Zustand $q \in Q$ führen wir zwei neue Zustände $(q, 0)$ und $(q, 1)$ ein, das heißt, die neue Zustandsmenge von $\mathcal{M}_{\text{even}}$ ist $Q \times \{0, 1\}$. Ist s der Startzustand von $\mathcal{M}$, dann ist $(s, 0)$ der neue Startzustand von $\mathcal{M}_{\text{even}}$. Wir können nun in den Zuständen speichern, ob bereits gerade oder ungerade viele Kopfbewegungen gemacht wurden.

Für einen Zustandsübergang $\delta(q, \mathsf{a}) = (q', \mathsf{a}', D)$ mit $q, q' \in Q$, $\mathsf{a}, \mathsf{a}' \in \Sigma$ und $D \in \{L, N, R\}$ in $\mathcal{M}$ fügen wir den Zustandsübergang $\delta((q, b), \mathsf{a}) = ((q', b'), \mathsf{a}', D)$ ein. Dabei ist $b = b'$, falls $D = N$ und sonst $b' = 1 - b$.

Die Menge der akzeptierenden Zustände in $\mathcal{N}$ ist $F \times \{0\}$, wobei F die Menge der akzeptierenden Zustände in $\mathcal{M}$ ist. Für jeden Zustand in $(q, 1) \in F \times \{1\}$ und jedes gelesene Zeichen wird ein Zustandsübergang nach $(q, 0)$ eingeführt, wobei eine Kopfbewegung nach links gemacht wird.

Die Abarbeitung eines Wortes ändert sich nicht, außer im letzten Schritt, in dem akzeptiert wird, falls sich die Turingmaschine in einem vormals akzeptierenden Zustand befindet. Also werden genau die gleichen Wörter akzeptiert. Außerdem befindet sich die Turingmaschine genau dann in einem Zustand $(q, 0)$, wenn sie gerade viele Kopfbewegungen gemacht hat. Da nur Zustände der Form $(q, 0)$ akzeptierend sind, werden alle Wörter die akzeptiert werden, nach gerade vielen Bewegungen akzeptiert.

(c) Wir nehmen an, L_{even} wird von einer Turingmaschine $\mathcal{M}_{\text{even}}$ entschieden. Damit konstruieren wir eine Turingmaschine $\mathcal{M}_u$, die L_u entscheidet. Für eine Eingabe $\mathcal{M}\#w$ konstruiert $\mathcal{M}_u$ die Turingmaschine $\mathcal{N}_{\mathcal{M}}$ wie in Teilaufgabe (b). Dann simuliert $\mathcal{M}_u$ die Turingmaschine $\mathcal{M}_{\text{even}}$ auf der Eingabe $\langle \mathcal{N}_{\mathcal{M}} \rangle \# w$ und übernimmt das Akzeptanzverhalten.

Dann gilt

$$w \in L(\mathcal{M}) \Leftrightarrow \mathcal{N}_{\mathcal{M}} \text{ akzeptiert } w \text{ nach einer geraden Anzahl von Kopfbewegungen}$$

$$\Leftrightarrow \langle \mathcal{N}_{\mathcal{M}} \rangle \# w \in L_{\text{even}} \, .$$

Da $\mathcal{M}_{\text{even}}$ immer hält, hält auch $\mathcal{M}_u$ auf jeder Eingabe und entscheidet somit L_u, was ein Widerspruch ist zur Unentscheidbarkeit der universellen Sprache. Also ist L_{even} nicht entscheidbar.

(d) Wäre das Komplement semi-entscheidbar, wäre auch L_{even} laut Vorlesung entscheidbar, was ein Widerspruch zu Teilaufgabe (c) wäre.