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

Hier Aufkleber mit Matrikelnummer anbringen

- 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 $\Sigma = \{\mathsf{0}, \mathsf{1}\}$, $V = \{S, X, Y\}$ und

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

(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.

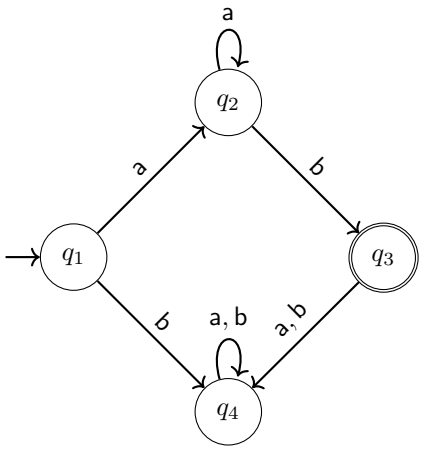
Nun erlauben wir Regeln der Form $X \to wY$ und $X \to \varepsilon$, wobei X, Y Variablen sind und $w \in \Sigma^*$ ein beliebiges Wort. Hierbei dürfen für unterschiedliche Regeln auch unterschiedliche Wörter gewählt werden. Solche Grammatiken nennen wir $*$-*rechtslinear*.

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

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.

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.

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:

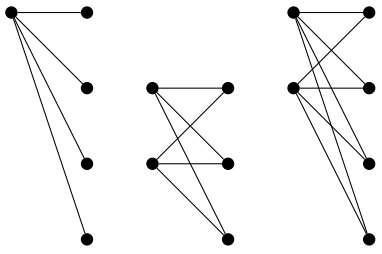
Die TUPLESPLITTING-Instanz $M = \{(1, 4), (2, 3), (2, 4)\}$ hat die Lösung $R = \{1, 3, 4\}, B = \{4, 2, 2\}$.

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:



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

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

(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:

[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.

(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.

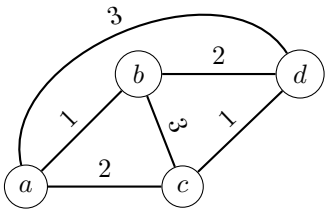
Problem 5: Approximation $2 + 5 + 3 = 10$ Punkte

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.

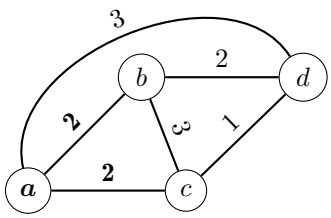
Edge Coloring	
Gegeben:	Ein endlicher Graph $G = (V, E)$
Gesucht:	Minimale Zahl an Farben, sodass alle Kanten gefärbt werden können und keine zwei Kanten mit gemeinsamem Endpunkt die gleiche Farbe haben.

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.

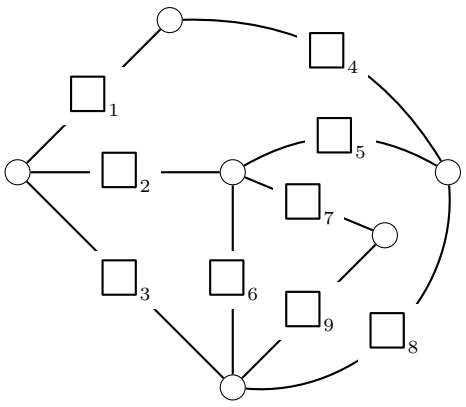
Wir sagen, eine Farbe i an einem Knoten v ist frei, wenn keine Kante an v schon mit Farbe i gefärbt ist.

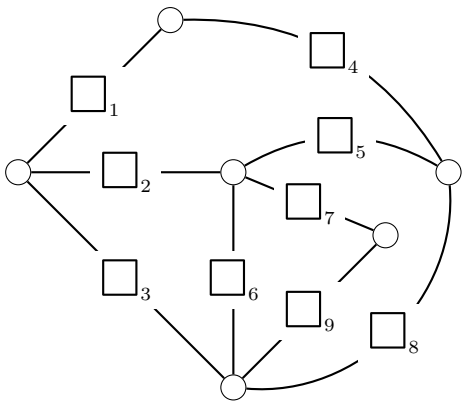
Algorithmus 1: Greedy Edge Coloring

```
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.





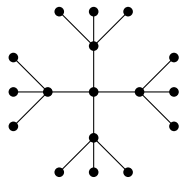
(b) Zeigen Sie, dass Greedy Edge Coloring ein polynomieller Approximationsalgorithmus mit relativer Gütegarantie von 2 für Edge Coloring ist.

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

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.