1. Klausur zur Vorlesung
Theoretische Grundlagen der Informatik
Wintersemester 2020/2021

Lösung!

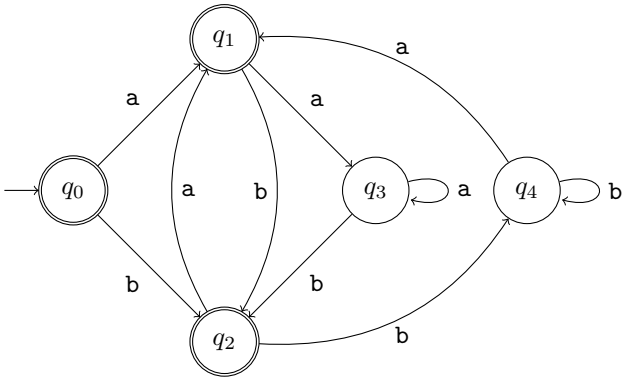
Beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen und Ihrer Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihrem Namen und 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.
- Es sind keine Hilfsmittel zugelassen.

	Mögliche Punkte					Erreichte Punkte				
	a	b	c	d	Σ	a	b	c	d	Σ
Aufg. 1	2	3	4	–	9				–	
Aufg. 2	4	4	–	–	8			–	–	
Aufg. 3	4	2	1	3	10					
Aufg. 4	2	3	3	–	8				–	
Aufg. 5	1	4	4	2	11					
Aufg. 6	4	4	–	–	8			–	–	
Aufg. 7	3	3	–	–	6			–	–	
Σ					60					

Problem 1: Reguläre Sprachen $2 + 3 + 4 = 9$ Punkte

Gegeben ist folgender endlicher Automat $\mathcal{A}$:



Dieser Automat erkennt die Sprache

$$L(\mathcal{A}) = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ endet nicht auf } \mathtt{aa} \text{ oder } \mathtt{bb}\}.$$

Für ein Alphabet Σ und ein Wort $w \in \Sigma^*$ bezeichnen wir mit $\text{pre}(w)$ die Menge aller Präfixe von w, außer w selbst.

Beispiel: $\text{pre}(\mathtt{POMMES}) = \{\varepsilon, \mathtt{P}, \mathtt{PO}, \mathtt{POM}, \mathtt{POMM}, \mathtt{POMME}\}$

Für eine Sprache $L \subseteq \Sigma^*$ sei $\text{pre}_{\text{all}}(L)$ die Menge aller Wörter aus L, für die alle Präfixe in L liegen. Formal ist

$$\text{pre}_{\text{all}}(L) = \{w \in L \mid \text{pre}(w) \subseteq L\}.$$

(a) Welche Sprache ergibt sich für $\text{pre}_{\text{all}}(L(\mathcal{A}))$?

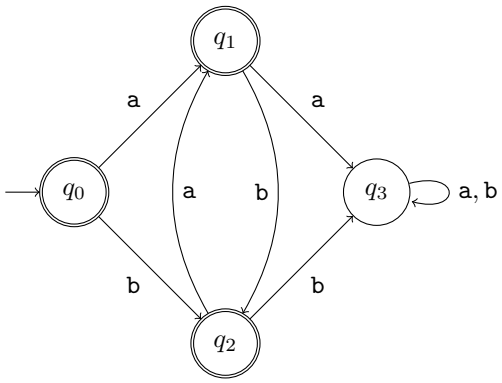
(b) Geben Sie einen endlichen Automaten an, der $\text{pre}_{\text{all}}(L(\mathcal{A}))$ erkennt.

(c) Zeigen Sie, dass die regulären Sprachen unter der Operation pre_{all} abgeschlossen sind.

Lösung:

(a) $\text{pre}_{\text{all}}(L(\mathcal{A})) = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ enthält nicht } \mathtt{aa} \text{ oder } \mathtt{bb}\}$

(b) Folgender DEA erkennt $\text{pre}_{\text{all}}(L(\mathcal{A}))$:



(c) Sei L eine beliebige reguläre Sprache und $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ ein DEA, der L akzeptiert. Verschmelze alle Zustände aus $Q \setminus F$ zu einem Müllzustand. Dadurch sind nur noch solche Pfade akzeptierend, die nur Endzustände besuchen. Also werden nur Wörter akzeptiert, für die alle Präfixe in L liegen.

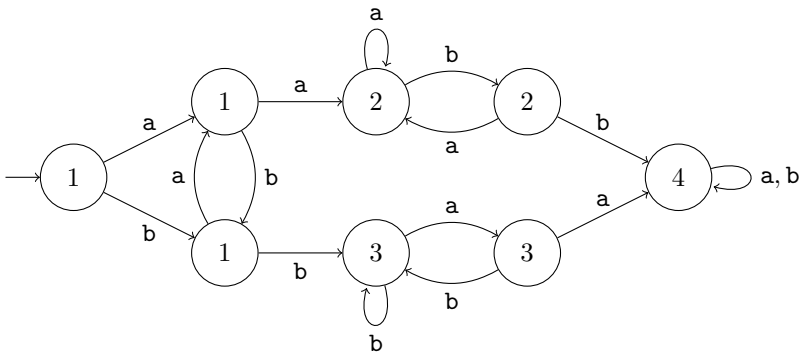
Problem 2: Endliche Klassifikatoren 4 + 4 = 8 Punkte

Endliche Automaten, wie wir sie aus der Vorlesung kennen, unterteilen Wörter in zwei Klassen: „akzeptiert“ und „nicht akzeptiert“. Deshalb werden sie auch *endliche Akzeptoren* genannt. In dieser Aufgabe wollen wir das Konzept zu *endlichen Klassifikatoren* erweitern, die Wörter in mehr als 2 Klassen unterteilen.

Gegeben seien ein Alphabet Σ und ein festes $k > 2$. Analog zum Konzept der formalen Sprache definieren wir eine *formale Klassifikation* als eine Abbildung $K : \Sigma^* \to \{1, \ldots, k\}$, die jedem Wort $w \in \Sigma^*$ eine *Klasse* $K(w)$ zuweist.

Ein *deterministischer endlicher Klassifikator* (DEK) ist ein 5-Tupel $\mathcal{A} = (Q, \Sigma, \delta, s, P)$ mit Zustandsmenge Q, Alphabet Σ, Überführungsfunktion δ und Startzustand s genau wie bei einem DEA. Statt einer Menge F von Endzuständen gibt es eine Zustandsfunktion $P : Q \to \{1, \ldots, k\}$, die jedem Zustand $q \in Q$ eine Klasse $P(q)$ zuweist. Ein endlicher Klassifikator $\mathcal{A}$ *erkennt* eine formale Klassifikation K, wenn er jedes Wort $w \in \Sigma^*$ in einen Zustand q mit $P(q) = K(w)$ überführt.

(a) Betrachten Sie folgenden DEK für $k = 4$. Die Zustände sind jeweils mit ihrer Klasse beschriftet.



Vervollständigen Sie die Definition der formalen Klassifikation K, die von diesem DEK erkannt wird.

$$K(w) = \begin{cases} 1, \text{ falls} \\ 2, \text{ falls} \\ 3, \text{ falls} \\ 4, \text{ falls} \end{cases}$$

Hinweis: Betrachten Sie die Teilwörter `aa` *und* `bb`*. Welche Zustände werden beim Lesen dieser Teilwörter erreicht?*

Zu einer formalen Klassifikation K definieren wir die Sprachen $L_1, \ldots, L_k$:

$$L_i = \{w \in \Sigma^* \mid K(w) = i\}$$

Also ist L_i die Sprache aller Wörter, die zur Klasse i gehören. Die L_i sind paarweise disjunkt und ihre Vereinigung entspricht Σ^*, d.h. jedes Wort ist in genau einer Sprache L_i enthalten.

(b) Zeigen Sie: Wenn es einen DEK gibt, der K erkennt, dann sind $L_1, \ldots, L_k$ alle regulär.

Lösung:

(a)

$$K(w) = \begin{cases} 1, \text{ falls } w \text{ weder } \texttt{aa} \text{ noch } \texttt{bb} \text{ enthält} \\ 2, \text{ falls } w \texttt{aa}, \text{ aber nicht } \texttt{bb} \text{ enthält} \\ 3, \text{ falls } w \texttt{bb}, \text{ aber nicht } \texttt{aa} \text{ enthält} \\ 4, \text{ falls } w \texttt{aa} \text{ und } \texttt{bb} \text{ enthält} \end{cases}$$

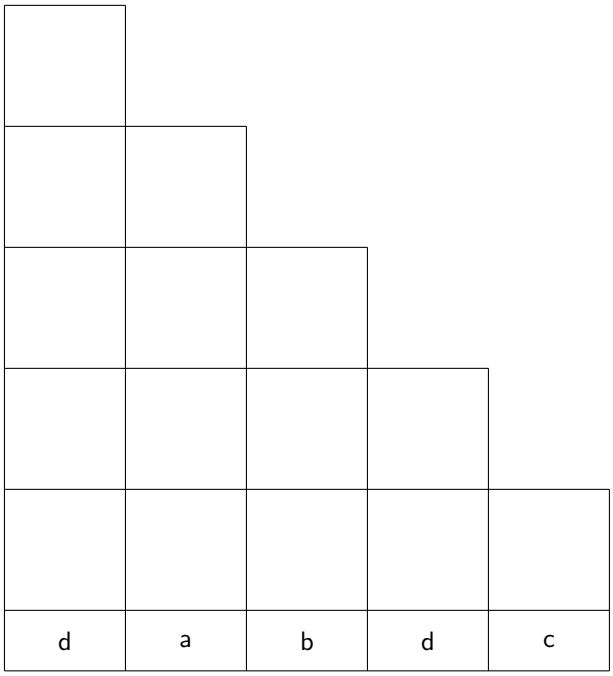
(b) Gegeben sei ein DEK $\mathcal{A} = (Q, \Sigma, \delta, s, P)$, der K erkennt. Betrachte nun für $i \in \{1, \ldots, k\}$ den endlichen Akzeptor $\mathcal{A}_i = (Q, \Sigma, \delta, s, F_i)$ mit $F_i = \{q \in Q \mid P(q) = i\}$. Offensichtlich überführt $\mathcal{A}_i$ jedes Wort $w \in \Sigma^*$ in denselben Zustand wie $\mathcal{A}$. Das Wort wird genau dann akzeptiert, wenn dieser Zustand zur Klasse i gehört. Da $\mathcal{A}$ ein endlicher Klassifikator für K ist, bedeutet das, dass w ebenfalls zur Klasse i gehört. Also akzeptiert $\mathcal{A}_i$ genau L_i.

Problem 3: CYK-Algorithmus $4 + 2 + 1 + 3 = 10$ Punkte

Gegeben sei die Grammatik $G = (\Sigma, V, S, R)$ mit Terminalen $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$, Nichtterminalen $V = \{S, A, B, C, D\}$, Startsymbol S und Produktionen

$$\begin{aligned} R = \{ & S \to CB \\ & A \to BA \mid CD \mid \texttt{a} \mid \texttt{d} \\ & B \to AC \mid \texttt{b} \\ & C \to DA \mid \texttt{b} \mid \texttt{c} \\ & D \to AB \mid \texttt{d}\}. \end{aligned}$$

(a) Überprüfen Sie mit dem CYK-Algorithmus, ob das Wort `dabdc` in der Sprache $L(G)$ enthalten ist.



(b) Es gibt zwei Ableitungsbäume für `dabdc` in G. Geben Sie beide an.

Wir führen nun Kosten für die Produktionen aus R ein.

- Produktionen mit Kosten 1: $A \to \texttt{a} \mid \texttt{d}$, $B \to \texttt{b}$, $C \to \texttt{b} \mid \texttt{c}$, $D \to \texttt{d}$
- Produktionen mit Kosten 2: $S \to CB$, $A \to CD$, $B \to AC$, $C \to DA$
- Produktionen mit Kosten 3: $A \to BA$, $D \to AB$

Wir definieren die Kosten eines Ableitungsbaums als die Summe der Kosten aller im Ableitungsbaum verwendeten Produktionen. Wenn eine Produktion mehrfach im Ableitungsbaum verwendet wird, zählt sie auch mehrfach. Die Ableitungskosten für ein Wort w definieren wir als das Minimum über die Kosten aller Ableitungsbäume, die w erzeugen.

(c) Berechnen Sie die Ableitungskosten von `dabdc`.

(d) Wie kann der CYK-Algorithmus so angepasst werden, dass er für das geprüfte Wort zusätzlich die Ableitungskosten berechnet?

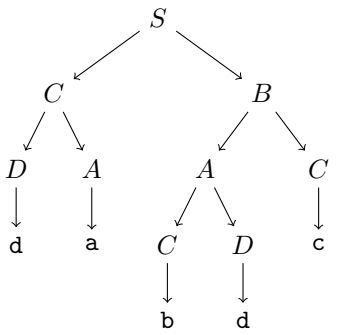
Lösung:

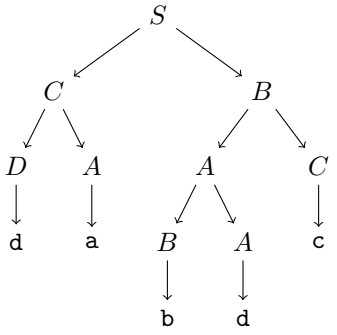
(a) Vergleiche folgende Tabelle:

S, D				
B, C	B, D			
S, D	A, C	S, B		
C	B, D	A	B	
A, D	A	B, C	A, D	C
d	a	b	d	c

Das Wort ist also in der Sprache enthalten.

(b) Es gibt zwei Ableitungsbäume:





(c) Der obere Baum hat Kosten 13. Der untere Baum hat Kosten 14. Also hat das Wort `dabdc` Ableitungskosten 13.

(d) Der ursprüngliche CYK-Algorithmus berechnet Variablenmengen V_{ij}, wobei V_{ij} alle Nichtterminale enthält, aus denen das Wort $w_i \dots w_j$ abgeleitet werden kann. Statt einzelner Nichtterminale X soll V_{ij} nun Tupel (X, c) enthalten, wobei c die minimalen Kosten sind, mit denen $w_i \dots w_j$ aus X abgeleitet werden kann.

Für die Mengen V_{ii} bestehen die Ableitungsketten jeweils aus einer einzelnen Produktion. Wenn es also eine Produktion $X \to w_i$ mit Kosten c gibt, fügen wir (X, c) in V_{ii} ein. Nun betrachten wir die Menge V_{ij} und eine Produktion $X \to YZ$ mit Kosten c. Falls es ein k gibt, sodass Tupel $(Y, c_Y) \in V_{ik}$ und $(Z, c_Z) \in V_{kj}$ existieren, so haben wir eine Ableitung mit Kosten $c_{\text{neu}} = c + c_Y + c_Z$ gefunden. Falls schon ein Tupel $(X, c_{\text{alt}}) \in V_{ij}$ existiert, ersetzen wir es durch $(X, \min(c_{\text{alt}}, c_{\text{neu}}))$. Ansonsten fügen wir das Tupel (X, c_{neu}) hinzu.

Falls das Eingabewort w in der Sprache enthalten ist, erzeugt der Algorithmus ein Tupel $(S, c_w) \in V_{1|w|}$, wobei c_w die Ableitungskosten von w sind.

Problem 4: Approximationsalgorithmen $2 + 3 + 3 = 8$ Punkte

Gegeben sei ein Minimierungsproblem mit ganzzahligen, positiven Lösungswerten (z.B. COLOR). Welche der folgende Aussagen sind richtig und welche falsch? Zeigen bzw. widerlegen Sie jeweils.

(a) Jeder Approximationsalgorithmus mit absoluter Gütegarantie 1 hat auch eine relative Gütegarantie von 2.

(b) Für jedes $c \geq 1$ gilt: Jeder Approximationsalgorithmus mit absoluter Gütegarantie c hat auch eine relative Gütegarantie von 2.

(c) Für jedes $c \geq 1$ gilt: Jeder Approximationsalgorithmus mit absoluter Gütegarantie c hat auch eine relative Gütegarantie von $c + 1$.

Lösung:

(a) Richtig. Da die Lösungswerte ganzzahlig und positiv sind, gilt $\mathrm{OPT}(I) \geq 1$. Also gilt:

$$\mathcal{A}(I) \leq \mathrm{OPT}(I) + 1 \leq 2 \cdot \mathrm{OPT}(I)$$

(b) Falsch. Betrachte z.B. $c = 3$ und eine Instanz I mit Optimalwert $\mathrm{OPT}(I) = 2$. Wenn ein Approximationsalgorithmus eine Lösung mit Wert $\mathcal{A}(I) = 5$ liefert, erfüllt das die absolute Gütegarantie, aber nicht die relative.

(c) Richtig. Analog zu Teilaufgabe (a) gilt:

$$\mathcal{A}(I) \leq \mathrm{OPT}(I) + c \leq \mathrm{OPT}(I) + c \cdot \mathrm{OPT}(I) = (c + 1) \cdot \mathrm{OPT}(I)$$

Problem 5: NP-Vollständigkeit $1 + 4 + 4 + 2 = 11$ Punkte

Wir betrachten Kreise in der Ebene. Zu einem Kreis gehört sowohl der Rand als auch das Innere. Wir sagen, dass sich zwei Kreise *schneiden*, wenn sie mindestens einen gemeinsamen Punkt haben. Insbesondere schneiden sich Kreise, wenn sie sich berühren, teilweise überlappen oder vollständig enthalten. Ein Kreis wird durch den Mittelpunkt und den Radius definiert.

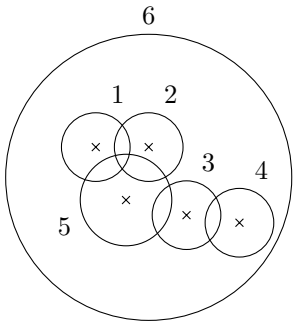


Abbildung 1: Kreise in der Ebene, die mit vier Farben gefärbt werden können, aber nicht mit drei Farben

Eine *Kreisfärbung* ordnet jedem Kreis eine Farbe zu, sodass je zwei Kreise, die sich schneiden, verschiedene Farben haben. Für eine natürliche Zahl k ist das k-FARBEN-KREISPROBLEM (k-FKP) wie folgt definiert:

Gegeben:	Menge von Kreisen in der Ebene
Frage:	Gibt es eine Kreisfärbung mit k Farben?

Sie dürfen voraussetzen, dass das 3-FARBEN-KREISPROBLEM NP-vollständig ist.

(a) Färben Sie die sechs Kreise in Abbildung 1 mit den Farben rot, blau, grün und gelb. Tragen Sie hierzu die verwendeten Farben in die Tabelle ein.

1	2	3	4	5	6
rot	blau	rot	blau	grün	gelb

(b) Gegeben sei eine Menge M von Kreisen in der Ebene. Konstruieren Sie eine Menge $M' \supseteq M$ von Kreisen, die genau dann eine Kreisfärbung mit $k+1$ Farben hat, wenn M eine Kreisfärbung mit k Farben hat. Begründen Sie, warum Sie genau eine Farbe mehr benötigen.

(c) Zeigen Sie, dass das 4-FARBEN-KREISPROBLEM NP-vollständig ist.

Hinweis: Sie dürfen die Aussage von Aufgabenteil (b) verwenden, auch wenn Sie (b) nicht gelöst haben.

(d) Beschreiben Sie, wie der Beweis so erweitert werden kann, dass er die NP-Vollständigkeit des k-FARBEN-KREISPROBLEMS für jedes $k \geq 3$ zeigt.

Lösung:

(b) M' besteht aus allen Kreisen aus M und einem weiteren Kreis, der alle Kreise aus M enthält. Wir zeigen,

$$M \text{ hat eine Kreisfärbung mit } k \text{ Farben} \iff M' \text{ hat eine Kreisfärbung mit } k+1 \text{ Farben.}$$

„⇒“ Verwende eine Kreisfärbung mit k Farben für M und eine weitere Farbe für den zusätzlichen Kreis. Es gibt also eine Kreisfärbung von M' mit $k+1$ Farben.

„⇐“ Betrachte eine Färbung ϕ von M' mit $k+1$ Farben. Der zusätzliche Kreis schneidet alle Kreise aus M und hat daher eine Farbe, die in M nicht vorkommt. Die Färbung ϕ eingeschränkt auf M ist also eine Färbung von M in k Farben.

(c) Um zu überprüfen, ob eine Lösung korrekt ist, prüfen wir für je zwei Kreise, ob sie sich schneiden und ob sie die gleiche Farbe haben. Außer dem zählen wir, wie viele Farben verwendet wurden. Dies ist in quadratischer Zeit möglich. Daher liegt das Problem in NP.

Wir reduzieren das 3-FARBEN-KREISPROBLEM auf das 4-FARBEN-KREISPROBLEM. Gegeben eine 3-FARBEN-KREISPROBLEM-Instanz M, konstruieren wir eine 4-FARBEN-KREISPROBLEM-Instanz M', sodass M genau dann eine Ja-Instanz ist, wenn M' eine Ja-Instanz ist.

Sei M eine 3-FARBEN-KREISPROBLEM-Instanz. Wie in Aufgabenteil (b) besteht M' aus allen Kreisen aus M und einem weiteren Kreis, der alle Kreise aus M enthält. Der zusätzliche Kreis kann in Polynomialzeit berechnet werden. Wähle zum Beispiel als Mittelpunkt den Mittelpunkt eines beliebigen Kreises aus M und als Radius den maximalen Abstand von zwei Kreismittelpunkten aus M plus den größten Radius aller Kreise aus M. M' kann in Polynomialzeit berechnet werden, da nur ein einziges Element hinzugefügt wird.

Nach Aufgabenteil (b) ist M genau dann 3-färbbar ist, wenn M' 4-färbbar ist.

(d) Mit den gleichen Argumenten wie in (c) liegt das k-FARBEN-KREISPROBLEM in NP. Wir zeigen die NP-Schwere mit Hilfe einer Induktion über k. Der Basisfall $k = 3$ wird nach Aufgabenstellung vorausgesetzt. Sei also das k-FARBEN-KREISPROBLEM NP-schwer für ein $k \geq 3$. Wir zeigen, dass das $(k+1)$-FARBEN-KREISPROBLEM ebenfalls NP-schwer ist. Sei M eine k-FARBEN-KREISPROBLEM-Instanz. Mit Aufgabenteil (b) konstruieren wir eine Menge M' von Kreisen, sodass M genau dann k-färbbar ist, wenn M' $(k+1)$-färbbar ist. Wie in (c) ist die Konstruktion polynomiell.

Problem 6: In-Place-Turingmaschine $4 + 4 = 8$ Punkte

Wir betrachten Turingmaschinen, deren Band auf die Felder beschränkt ist, auf denen die Eingabe steht. Zusätzlich gibt es vor dem ersten und nach dem letzten Symbol der Eingabe ein Blanksymbol, das aber nicht überschrieben werden darf. Die Turingmaschine darf auf den Feldern der Eingabe lesen und schreiben, aber keinen weiteren Platz verwenden. Der Platzbedarf einer Turingmaschine bezeichnet die Anzahl der Felder, auf denen im Laufe der Abarbeitung (mindestens einmal) geschrieben wird. Eine solche Turingmaschine nennen wir eine *In-Place-Turingmaschine*.

Sie dürfen in dieser Aufgabe verwenden, dass

- die universelle Sprache semi-entscheidbar ist, aber nicht kontextsensitiv,
- die Sprache $\{w\#w \mid w \in \{0,1\}^*\}$ kontextsensitiv ist, aber nicht kontextfrei und
- die Sprache $\{w\#w^R \mid w \in \{0,1\}^*\}$ kontextfrei ist, aber nicht regulär.

(a) Zeigen Sie, dass es eine Sprache gibt, die von einer In-Place-Turingmaschine erkannt wird, aber von keinem Kellerautomaten.

Hinweis 1: Wählen Sie eine der angegebenen Sprachen und beschreiben Sie, wie eine In-Place-Turingmaschine diese Sprache akzeptiert. Begründen Sie, warum Ihre Wahl die Aufgabe löst.

Hinweis 2 (falls Sie nicht wissen, welche Sprache Sie wählen müssen): Sie erhalten Teilpunkte, wenn Sie für eine beliebige der angegebenen Sprachen zeigen, wie sie von einer In-Place-Turingmaschine akzeptiert wird.

(b) Zeigen Sie, dass In-Place-Turing-Maschinen weniger mächtig sind als klassische Turingmaschinen (d.h. 1 Kopf, 1 Band, wie in der Vorlesung definiert). Zeigen Sie hierfür, dass

- jede Sprache, die von einer In-Place-Turingmaschine akzeptiert wird, auch von einer klassischen Turingmaschine akzeptiert wird und
- es eine Sprache gibt, die von einer klassischen Turingmaschine akzeptiert wird, aber von keiner In-Place-Turingmaschine.

Hinweis: Kennen Sie ein Maschinenmodell, dass zwischen In-Place-Turingmaschinen und klassischen Turingmaschinen liegt?

Lösung:

(a) Wir konstruieren eine In-Place-Turingmaschine, die die Sprache $\{w\#w \mid w \in \{0,1\}^*\}$ akzeptiert. Kellerautomaten können genau die kontextfreien Sprachen akzeptieren. Nach Aufgabenstellung ist diese Sprache nicht kontextfrei, wird also von keinem Kellerautomaten akzeptiert. Lies das erste Symbol, ersetze es durch ein Blanksymbol und merke das Symbol mit Hilfe von Zuständen. Laufe zum ersten Feld nach # (später *) und überprüfe, ob das Symbol mit dem gemerkten Symbol übereinstimmt. Ersetze dabei das gelesene Symbol durch *. Laufe zurück zum Anfang (erstes Nicht-Blanksymbol) und wiederhole. Falls ein unerwartetes Symbol gelesen wird (nicht das gemerkte oder weder 0 noch 1), lehne ab. Wenn # als erstes Nicht-Blank-Symbol gelesen wird (d.h. das erste Wort abgearbeitet ist), laufe zum Ende der Eingabe und überprüfe, ob auch das zweite Wort abgearbeitet ist und akzeptiere entsprechend oder lehne ab.

(b) Eine In-Place-Turingmaschine ist insbesondere eine klassische Turingmaschine und kann daher nicht mehr Sprachen akzeptieren als letztere. Jede Sprache, die von einer In-Place-Turingmaschine akzeptiert wird, wird auch von einer $\mathcal{NTAPE}(n)$-Turingmaschine akzeptiert. Kopiere hierzu die Eingabe auf das Arbeitsband und simuliere dort die In-Place-Turingmaschine. Da $\mathcal{NTAPE}(n)$-Turingmaschinen genau die kontextsensitiven Sprachen akzeptieren, ist jede Sprache, die von einer In-Place-Turingmaschine akzeptiert wird, kontextsensitiv. Nach Aufgabenstellung gibt es aber eine

Sprache, die semi-entscheidbar ist, d.h. von einer klassischen Turingmaschine akzeptiert wird, aber nicht kontextsensitiv ist.

Problem 7: Entscheidbarkeit 3 + 3 = 6 Punkte

Wir betrachten die folgende Sprache

$$L = \{(\langle\mathcal{M}\rangle, q, w) \mid \langle\mathcal{M}\rangle \text{ ist die Gödelnummer einer Turingmaschine } \mathcal{M} \text{ mit Zustandsmenge } Q \text{ und Eingabealphabet } \Sigma, q \in Q, w \in \Sigma^*, \mathcal{M} \text{ geht bei Eingabe } w \text{ mindestens einmal in den Zustand } q\}.$$

(a) Gegeben sei eine Turingmaschine $\mathcal{M}$ und ein Wort w. Konstruieren Sie eine Turingmaschine $\mathcal{M}'$ mit einem Zustand q' und ein Wort w', sodass

$$\mathcal{M}' \text{ bei Eingabe } w' \text{ in Zustand } q' \text{ geht } \iff \mathcal{M} \text{ auf der Eingabe } w \text{ hält.}$$

(b) Zeigen Sie, dass L nicht entscheidbar ist. Verwenden Sie nicht den Satz von Rice!

Hinweis 1: Nehmen Sie an, dass es einen Entscheider für L gibt und bauen Sie daraus einen Entscheider für das Halteproblem.

Hinweis 2: Sie dürfen die Aussage aus (a) verwenden, auch wenn Sie (a) nicht gelöst haben.

Lösung:

(a) Sei $\mathcal{M}'$ eine Turingmaschine, für deren Eingabe w' wir $(\mathcal{M}, w)$ wählen. $\mathcal{M}'$ simuliert $\mathcal{M}$ mit Eingabe w. Falls $\mathcal{M}$ hält, geht $\mathcal{M}'$ in einen Zustand q', der für die Simulation nicht verwendet wird. Falls $\mathcal{M}$ nicht hält, so hält auch $\mathcal{M}'$ nicht und der Zustand q' wird nicht verwendet. Also geht $\mathcal{M}'$ bei Eingabe w' genau dann in Zustand q', wenn $\mathcal{M}$ auf der Eingabe w hält.

(b) Wir nehmen an, L wäre entscheidbar, und zeigen, dass dann das Halteproblem ebenfalls entscheidbar wäre. Sei $\mathcal{M}_L$ eine Turingmaschine, die L entscheidet. Wir bauen eine Turingmaschine $\mathcal{M}_H$, die das Halteproblem entscheidet. Sei $(\langle\mathcal{M}\rangle, w)$ eine Eingabe für $\mathcal{M}_H$. $\mathcal{M}_H$ konstruiert wie in (a) das Tupel $(\langle\mathcal{M}'\rangle, q', w')$ und übergibt dies als Eingabe an $\mathcal{M}_L$. $\mathcal{M}_H$ simuliert $\mathcal{M}_L$ und gibt die Antwort von $\mathcal{M}_L$ aus. Nach (a) ist $(\langle\mathcal{M}'\rangle, q', w')$ genau dann in L, wenn $\mathcal{M}$ bei Eingabe w hält. Da $\mathcal{M}_L$ in jedem Falle hält, entscheidet $\mathcal{M}_H$ das Halteproblem.

Über das Halteproblem wissen wir allerdings, dass es nicht entscheidbar ist, also haben wir einen Widerspruch und folgern, dass L ebenfalls nicht entscheidbar ist.