2. Klausur zur Vorlesung
Theoretische Grundlagen der Informatik
Wintersemester 2021/2022

Lösung!

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

Problem 1: Grammatiken/Pumping-Lemma $1 + 1 + 3 + 3 + 1 = 9$ Punkte

Eine *lineare Grammatik* ist eine kontextfreie Grammatik mit der zusätzlichen Eigenschaft, dass auf der rechten Seite jeder Regel höchstens eine Variable vorkommt. Wie immer bei kontextfreien Grammatiken besteht die linke Seite jeder Regel aus genau einer Variable. Eine *lineare Sprache* ist eine Sprache, die von einer linearen Grammatik erzeugt wird.

Wir wollen in dieser Aufgabe die Klasse der linearen Sprachen untersuchen. Dazu betrachten wir zwei Beispielsprachen:

- $L_1 = \{\mathsf{a}^j\mathsf{b}^j \mid j \in \mathbb{N}_0\}$. Aus der Vorlesung ist bekannt, dass L_1 kontextfrei, aber nicht regulär ist.
- $L_2 = \{\mathsf{a}^j\mathsf{b}^j\mathsf{c}^k\mathsf{d}^k \mid j, k \in \mathbb{N}_0\}$

(a) Geben Sie eine lineare Grammatik für L_1 an.

(b) Geben Sie eine kontextfreie Grammatik für L_2 an.

Für lineare Sprachen kann folgendes Pumping-Lemma gezeigt werden:

> Sei L eine lineare Sprache. Dann existiert eine Zahl $n \in \mathbb{N}$, sodass für jedes Wort $z \in L$ mit $|z| > n$ eine Darstellung
>
> $$z = uvwxy \text{ mit } |uvxy| \leqslant n, vx \neq \varepsilon$$
>
> existiert, bei der auch $uv^iwx^iy \in L$ ist für alle $i \in \mathbb{N}_0$.

(c) Zeigen Sie (durch explizites Nachrechnen), dass L_1 die Aussage des Pumping-Lemmas für lineare Sprachen erfüllt.

(d) Zeigen Sie, dass L_2 nicht linear ist.

(e) Wir bezeichnen mit $\mathcal{L}_i$ die Menge der Sprachen von Chomsky-Typ i und mit $\mathcal{L}_{\text{lin}}$ die Menge der linearen Sprachen. In welcher Beziehung stehen $\mathcal{L}_2$, $\mathcal{L}_3$ und $\mathcal{L}_{\text{lin}}$ zueinander? Kreuzen Sie die (einzige) richtige Wahl an. Begründen Sie für die richtige Wahl kurz beide Inklusionen bzw. (Un-)Gleichheiten.

- ☐ $\mathcal{L}_3 = \mathcal{L}_{\text{lin}} \subsetneq \mathcal{L}_2$
- ☐ $\mathcal{L}_3 \subsetneq \mathcal{L}_{\text{lin}} \subsetneq \mathcal{L}_2$
- ☐ $\mathcal{L}_{\text{lin}} \subseteq \mathcal{L}_3 \subseteq \mathcal{L}_2$
- ☐ $\mathcal{L}_3 \subsetneq \mathcal{L}_{\text{lin}} = \mathcal{L}_2$

Lösung:

(a) $G_1 = (\{\mathsf{a}, \mathsf{b}\}, \{S\}, S, \{S \to \mathsf{a}S\mathsf{b} \mid \varepsilon\})$

(b) $G_2 = (\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}, \{S, X, Y\}, S, R)$ mit

$$\begin{aligned} R = \{&S \to XY \\ &X \to \mathsf{a}X\mathsf{b} \mid \varepsilon \\ &Y \to \mathsf{c}Y\mathsf{d} \mid \varepsilon\} \end{aligned}$$

(c) Wähle $n = 2$. Wörter aus L_1 der Länge > 2 haben die Form $\mathsf{a}^j\mathsf{b}^j$ mit $j > 1$. Wähle die Zerlegung $z = uvwxy$ mit $u = y = \varepsilon$, $v = \mathsf{a}$, $x = \mathsf{b}$ und $w = \mathsf{a}^{j-1}\mathsf{b}^{j-1}$. Dann gilt $|uvxy| = 2$ und $vx \neq \varepsilon$. Für $i \in \mathbb{N}_0$ gilt $uv^iwx^iy = \mathsf{a}^{j+(i-1)}\mathsf{b}^{j+(i-1)} \in L$.

(d) Sei $n \in \mathbb{N}$. Wähle das Wort $z = \mathsf{a}^n\mathsf{b}^n\mathsf{c}^n\mathsf{d}^n$ mit $|z| > n$. Dann gilt für jede Zerlegung $z = uvwxy$ mit $|uvxy| \leqslant n$, dass w alle b's und c's enthalten muss. Aus $vx \neq \varepsilon$ folgt, dass vx mindestens ein a oder d enthält. Das bedeutet, dass uv^2wx^2y mindestens ein a oder d mehr enthält als w, aber genauso viele b's und c's. Also gilt $uv^2wx^2y \notin L_2$.

(e) $\mathcal{L}_3 \subsetneq \mathcal{L}_{\text{lin}} \subsetneq \mathcal{L}_2$

Begründung:

- $\mathcal{L}_3 \subseteq \mathcal{L}_{\text{lin}}$: Rechtslineare Grammatiken erfüllen alle Eigenschaften von linearen Grammatiken.
- $\mathcal{L}_3 \neq \mathcal{L}_{\text{lin}}$: L_1 ist nicht regulär, aber nach (**??**) linear.
- $\mathcal{L}_{\text{lin}} \subseteq \mathcal{L}_2$: Lineare Grammatiken sind als Spezialform von kontextfreien Grammatiken definiert.
- $\mathcal{L}_{\text{lin}} \neq \mathcal{L}_2$: L_2 ist nach (**??**) kontextfrei, aber nach (**??**) nicht linear.

Problem 2: Grammatiken/Automaten $3 + 2 + 5 = 10$ Punkte

Eine Grammatik $G = (\Sigma, V, S, R)$ heißt *fast-rechtslinear*, falls ihre Regeln folgende Form haben. Regeln ohne Startsymbol sind rechtslinear, haben also die Form

$$(i) \qquad Y \to wZ \mid \varepsilon \quad \text{mit } w \in \Sigma \text{ und } Y, Z \in V \setminus \{S\}.$$

Außerdem ist **genau** eine Regel erlaubt, die das Startsymbol auf eine oder mehrere Variablen abbildet. Diese Regel hat also die Form

$$(ii) \qquad S \to \alpha \qquad \text{mit } \alpha \in (V \setminus \{S\})^+.$$

Dabei darf niemals auf das Startsymbol abgebildet werden.

(a) Die Grammatik $G_{\text{Bsp}} = (\Sigma, V, S, R)$ mit $\Sigma = \{0, 1\}$, $V = \{S, A, B, C\}$ und folgender Regelmenge ist fast-rechtslinear.

$$\begin{aligned} R = \{ \;& S \to ACA \\ & A \to 0B \mid \varepsilon \\ & B \to 0A \\ & C \to 1C \mid \varepsilon \\ \} \end{aligned}$$

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

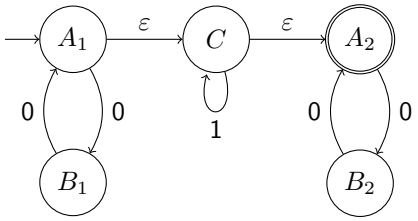
Wir zeigen nun, dass fast-rechtslineare Grammatiken genau die regulären Sprachen erzeugen.

(b) Zeigen Sie, dass jede reguläre Sprache von einer fast-rechtslinearen Grammatik erzeugt werden kann.

(c) Sei G eine fast-rechtslineare Grammatik. Geben Sie einen endlichen Automaten an, der $L(G)$ erkennt.

Lösung:

(a)



Der Automat besteht aus drei Teilautomaten (zwei Mal der Automat für die zweite und dritte Ableitungsregeln, einmal der für die vierte), die entsprechend der ersten Regel mit Epsilonübergängen verbunden sind.

(b) Sei L eine reguläre Sprache. Dann existiert eine rechtslineare Grammatik $G = (\Sigma, V, S, R)$ mit $L(G) = L$. Wir definieren eine äquivalente fast-rechtslineare Grammatik $G' = (\Sigma, V', S', R')$ wie folgt. Die Variablenmenge sei $V' = V \cup \{S'\}$, wobei das Startsymbol S' eine neue Variable ist, die nicht in V enthalten ist. Die Regelmenge R' sei R mit der zusätzlichen Regel $S' \to S$. Da S' nie auf der rechten Seite einer Regel vorkommt, ist G' fast-rechtslinear. Für jede Ableitung $S \overset{*}{\to}_G w$ existiert die Ableitung $S' \to S \overset{*}{\to}_{G'} w$ und umgekehrt, daher erzeugen G und G' die gleiche Sprache.

(c) Sei nun $G = (\Sigma, V, S, R)$ eine fast-rechtslineare Grammatik. Wir betrachten zunächst die Regel $S \to \alpha$ mit $\alpha = X_1 \dots X_n$ aus R. Für $i = 1, \dots, n$ definieren wir die Grammatik $G_i = (\Sigma, V \setminus \{S\}, X_i, R_i)$, wobei R_i alle Regeln aus R ohne Startsymbol enthält. Da die einzige Regel in R, die nicht rechtslinear ist, das Startsymbol enthält, ist G_i rechtslinear. Nach Vorlesung gibt es einen endlichen Automaten $\mathcal{A}_i$, der $L(G_i)$ erkennt. Wir verbinden nun die Automaten $\mathcal{A}_1, \dots, \mathcal{A}_n$ zu einem Automaten $\mathcal{A}$, der $L(G)$ erkennt. Der Startzustand von $\mathcal{A}$ ist der Startzustand von $\mathcal{A}_1$ und für die akzeptierenden Zustände verwenden wir die akzeptierenden Zustände von $\mathcal{A}_n$. Außerdem führen wir für $i = 1, \dots, n-1$ einen ε-Übergang von jedem akzeptierenden Zustand von $\mathcal{A}_i$ zum Startzustand von $\mathcal{A}_{i+1}$ ein.

Problem 3: NP-Vollständigkeit $1 + 1 + 1 + 6 = 9$ Punkte

Sei U eine Menge von Variablen und C eine Menge von Klauseln über U. In dieser Aufgabe darf jedes Literal höchstens einmal pro Klausel vorkommen. Aus der Vorlesung kennen Sie folgende Definition einer erfüllenden Wahrheitsbelegung.

Eine Wahrheitsbelegung ist *erfüllend* für (U, C), wenn jede Klausel ein wahres Literal enthält.

Wir definieren nun einen abgewandelten Erfüllbarkeitsbegriff.

Eine Wahrheitsbelegung ist NAE-*erfüllend* für (U, C), wenn jede Klausel ein wahres Literal und ein falsches Literal enthält.

Wir betrachten das folgende NP-schwere Entscheidungsproblem NOTALLEQUAL (NAE):

NOTALLEQUAL (NAE)	
Gegeben:	Menge U von Variablen Menge C von Klauseln über U
Frage:	Existiert NAE-erfüllende Wahrheitsbelegung für (U, C)?

Die folgende Variante von NAE verbietet negierte Variablen in den Klauseln, das heißt für jede Variable $u \in U$ darf das Literal u in den Klauseln vorkommen, aber das negierte Literal $\bar{u}$ darf in keiner Klausel vorkommen.

MONOTONENOTALLEQUAL (MNAE)	
Gegeben:	Menge U von Variablen Menge C von Klauseln über U, wobei kein Literal negiert ist
Frage:	Existiert NAE-erfüllende Wahrheitsbelegung für (U, C)?

(a) Geben Sie eine Lösung der folgenden MNAE-Instanz an, indem Sie die angegebene Tabelle ausfüllen.

$$U = \{a, b, c, d, e\}$$
$$C = \{(a, d), (b, e), (a, b, c), (b, c, d), (a, c, e)\}$$

a	b	c	d	e
`wahr`	`wahr`	`falsch`	`falsch`	`falsch`

(b) Sei $U = \{u, u'\}$ eine Menge von Variablen. Konstruieren Sie eine Menge C von Klauseln ohne negierte Literale über U, sodass für jede NAE-erfüllende Wahrheitsbelegung $t \colon U \to \{\texttt{wahr}, \texttt{falsch}\}$ gilt:

$$t(u) = \texttt{wahr} \iff t(u') = \texttt{falsch}.$$

(c) Zeigen Sie, dass MNAE in NP liegt. Geben Sie dazu an, woraus ein Lösungsvorschlag für eine MNAE-Instanz (U, C) besteht. Geben Sie außerdem die Laufzeit der Überprüfung in Abhängigkeit von $|U|$ und $|C|$ an, wobei $|M|$ die Anzahl der Elemente einer Menge M bezeichnet.

(d) Zeigen Sie, dass MONOTONENOTALLEQUAL NP-schwer ist.

Hinweis: Falls Sie Aufgabenteil **??** *nicht gelöst haben, dürfen Sie die Notation $f(u, u')$ für eine Klauselmenge mit der angegebenen Eigenschaft verwenden.*

Lösung:

(b) $C = \{(u, u')\}$

(c) Eine Lösung einer MNAE-Instanz (U, C) besteht aus einer Wahrheitsbelegung $t\colon U \to \{\texttt{wahr}, \texttt{falsch}\}$. Zur Überprüfung einer Lösung iterieren wir über alle Klauseln und für jede Klausel über alle vorkommenden Literale und prüfen, ob in jeder Klausel mindestens ein Literal auf `wahr` abgebildet wird und mindestens eines auf `falsch`. Falls dies für eine Klausel nicht der Fall ist, lehnen wir ab, sonst akzeptieren wir, nachdem alle Klauseln überprüft sind. Dabei werten wir t höchstens $|C| \cdot |U|$ mal aus, die Überprüfung ist also polynomiell in der Eingabegröße.

(d) Wir zeigen nun die NP-Schwere mit Hilfe einer polynomiellen Reduktion von NAE zu MNAE. Sei (U, C) eine NAE-Instanz. Wir konstruieren eine MNAE-Instanz (U', C') wie folgt. Die Variablenmenge U' besteht aus U und einer Kopie von U, d.h. $U' = U \cup \{u' \mid u \in U\}$. Für die Klauselmenge C' verwenden wir die Klauselmenge C, wobei jedes negierte Literal $\bar{u}$ durch u' ersetzt wird. Zusätzlich fügen wir für jede Variable $u \in U$ die Klausel (u, u') zu C' hinzu.

Die Konstruktion von U' ist in Zeit $\mathcal{O}(|U|)$ möglich, die Konstruktion von C' in $\mathcal{O}(|C| \cdot |U|)$ für das Ändern der Klauselmenge C plus $\mathcal{O}(|U|)$ für die Konstruktion der zusätzlichen Klauseln. Insgesamt ist die Reduktion also polynomiell in der Eingabegröße.

Wir zeigen nun, dass (U, C) genau dann eine Ja-Instanz von NAE ist, wenn (U', C') eine Ja-Instanz von MNAE ist.

$\Longrightarrow$ Sei $t\colon U \to \{\texttt{wahr}, \texttt{falsch}\}$ eine NAE-erfüllende Wahrheitsbelegung für (U, C). Wir definieren eine Wahrheitsbelegung $t'\colon U' \to \{\texttt{wahr}, \texttt{falsch}\}$ durch

$$t'(u) = \begin{cases} t(u) & \text{falls } u \in U \\ \neg t(u) & \text{falls } u \notin U. \end{cases}$$

Wir zeigen, dass t' eine NAE-erfüllende Wahrheitsbelegung für (U', C') ist. Für Klauseln, die aus C entstanden sind, entspricht die Wahrheitsbelegung t' der Wahrheitsbelegung t. Da t NAE-erfüllend für (U, C) ist, gilt dies auch für die entsprechenden Klauseln in C'. Falls $t'(u)$ auf `wahr` gesetzt wird, so wird $t'(u')$ auf `falsch` gesetzt und umgekehrt. Also enthalten Klauseln der Form (u, u') ein wahres und ein falsches Literal. Wir folgern, dass t' NAE-erfüllend für (U', C') ist, und somit, dass (U', C') eine Ja-Instanz von MNAE ist.

$\Longleftarrow$ Sei nun $t'\colon U' \to \{\texttt{wahr}, \texttt{falsch}\}$ eine NAE-erfüllende Wahrheitsbelegung für (U', C') und sei t die Wahrheitsbelegung t' eingeschränkt auf U. Wir zeigen, dass t NAE-erfüllend für (U, C) ist. Nach **??** gilt für jedes $u \in U$, dass $t'(u) = \texttt{wahr} \iff t'(u') = \texttt{falsch}$. Mit $t(u) = t'(u)$ für $u \in U$ folgt $t(u) = \texttt{wahr} \iff t'(u') = \texttt{falsch}$, also wird $\bar{u}$ mit t zum gleichen Wert ausgewertet wie u' mit t'. Die Wahrheitsbelegung der Klauseln in C entspricht also der für C', also ist t ebenfalls NAE-erfüllend und (U, C) eine Ja-Instanz.

Problem 4: Approximation $1 + 5 + 2 = 8$ Punkte

Wir betrachten in dieser Aufgabe ein beliebiges Minimierungsproblem Π.

(a) Sei $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ein PTAS für Π. Zeigen Sie: Für jede Konstante $\varepsilon > 0$ existiert ein polynomieller Approximationsalgorithmus mit relativer Gütegarantie $1 + \varepsilon$.

Sei nun $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ein FPTAS für Π. Sei außerdem p ein Polynom, sodass für alle Instanzen I von Π gilt: $\mathrm{OPT}(I) \leqslant p(|I|)$.

(b) Zeigen Sie: Für jede Konstante $\delta > 0$ existiert ein polynomieller Approximationsalgorithmus mit absoluter Gütegarantie δ.

(c) Für eine Instanz I von Π und eine (nicht notwendigerweise optimale) Lösung L von I sei $w(L)$ der Lösungswert von L. Wir nehmen nun zusätzlich an, dass es eine Konstante $c > 0$ gibt, sodass für jede Instanz I von Π und alle Lösungen L, L' von I mit $w(L) \neq w(L')$ gilt: $|w(L) - w(L')| \geqslant c$. Zeigen Sie, dass dann $\Pi \in \mathsf{P}$ gilt.

Lösung:

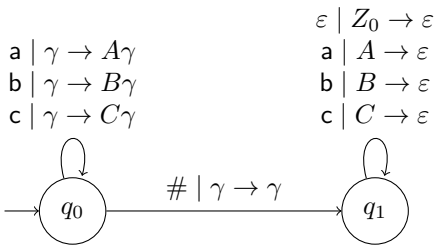
(a) Nach Definition ist die Laufzeit von $\mathcal{A}_\varepsilon$ polynomiell in der Eingabegröße und ε ist eine Konstante.

(b) Betrachte folgenden Algorithmus: Für eine gegebene Instanz I, berechne $\varepsilon = \delta / p(|I|)$ und führe $\mathcal{A}_\varepsilon$ auf I aus. Der Algorithmus ist polynomiell in $|I|$ und $1/\varepsilon = p(|I|)/\delta$, also polynomiell in $|I|$. Für den Lösungswert ergibt sich $\mathcal{A}_\varepsilon(I) \leqslant (1 + \varepsilon)\,\mathrm{OPT}(I) = \mathrm{OPT}(I) + \varepsilon\,\mathrm{OPT}(I) \leqslant \mathrm{OPT}(I) + \varepsilon p(|I|) = \mathrm{OPT}(I) + \delta$. Also hat der Algorithmus absolute Gütegarantie δ.

(c) Führe den Algorithmus aus (**??**) für $\delta = \frac{c}{2}$ aus. Dann gilt $\mathcal{A}_\varepsilon(I) \leqslant \mathrm{OPT}(I) + \frac{c}{2} < \mathrm{OPT}(I) + c$. Für jede Lösung L' von I mit $w(L') > \mathrm{OPT}(I)$ gilt $w(L') - \mathrm{OPT}(I) \geqslant c$ und damit $\mathcal{A}_\varepsilon(I) < \mathrm{OPT}(I) + (w(L') - \mathrm{OPT}(I)) = w(L')$. Also muss $\mathcal{A}_\varepsilon(I) = \mathrm{OPT}(I)$ gelten, d.h. $\mathcal{A}_\varepsilon$ löst Π exakt in Polynomialzeit.

Problem 5: Kellerautomaten $1 + 2 + 6 = 9$ Punkte

Betrachten Sie den Kellerautomaten $\mathcal{M}_1 = (Q_1, \Sigma, \Gamma_1 = \{Z_0, A, B, C\}, q_0, Z_0, \delta_1)$ über dem Eingabealphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \#\}$, der durch leeren Stack akzeptiert und folgenden Übergangsgraphen besitzt:



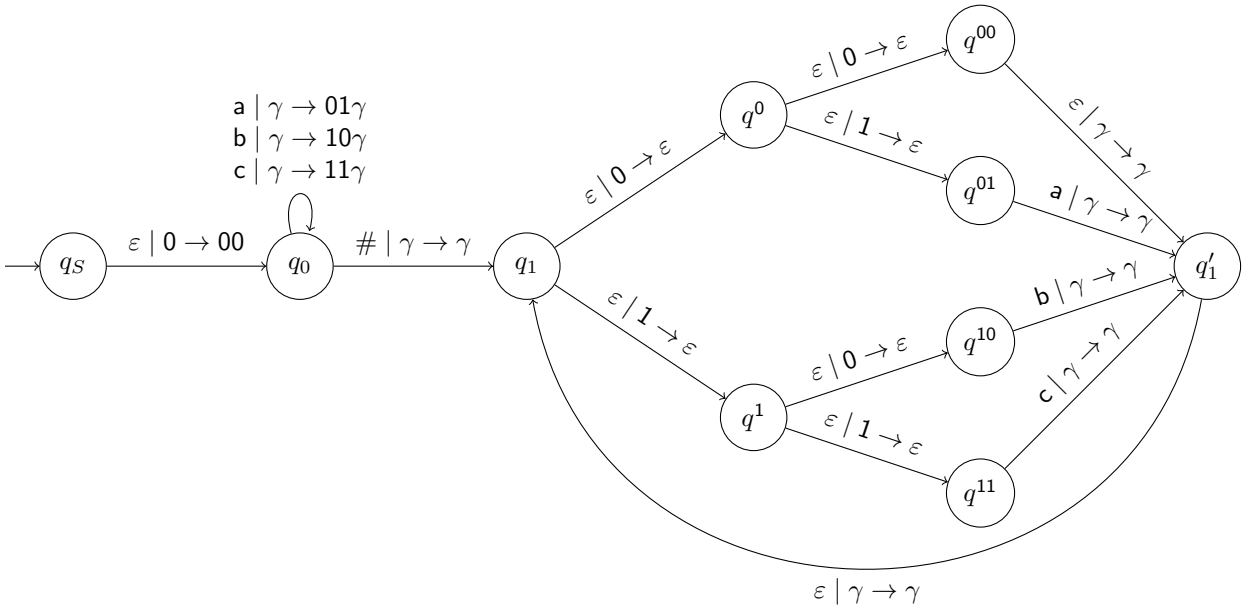
Das Symbol γ steht dabei für ein beliebiges Symbol $\gamma \in \Gamma_1$.

Wie aus der Übung bekannt, bedeutet die Notation $\mathsf{a} \mid B \to AB$: Beim zugehörigen Zustandsübergang wird das Eingabesymbol a gelesen, das oberste Stacksymbol B vom Stack gelöscht und AB auf den Stack gelegt. Dabei ist das linkeste Zeichen (hier also A) das oberste Zeichen auf dem Stack.

(a) Welche Sprache erkennt der Kellerautomat $\mathcal{M}_1$?

Im Folgenden betrachten wir *0-1-Kellerautomaten*. Das sind nichtdeterministische Kellerautomaten, deren Stackalphabet $\Gamma_2 = \{0, 1\}$ nur aus zwei Symbolen besteht. Das initiale Stacksymbol ist 0.

Wir definieren einen 0-1-Kellerautomaten, der die gleiche Sprache wie $\mathcal{M}_1$ erkennt. Dies ist der Automat $\mathcal{M}_2 = (Q_2, \Sigma, \Gamma_2 = \{0, 1\}, q_S, 0, \delta_2)$, der ebenfalls durch leeren Stack akzeptiert und folgenden Übergangsgraphen besitzt:



Auch hier steht das Symbol γ für ein beliebiges Symbol $\gamma \in \Gamma_2$.

(b) Geben Sie die Zustandsfolge an, die $\mathcal{M}_2$ bei der Abarbeitung von $w = \mathsf{ab}\#\mathsf{ba}$ durchläuft. Geben Sie zusätzlich den Stackinhalt von $\mathcal{M}_2$ an, wenn er zum ersten Mal in den Zustand q_1 übergeht.

(c) Zeigen Sie, dass nichtdeterministische Kellerautomaten und 0-1-Kellerautomaten gleich mächtig sind.

Hinweis: Geben Sie dazu unter anderem eine geeignete Kodierung für die Stacksymbole in Γ *an und beschreiben Sie, wie ein 0-1-Kellerautomat ein kodiertes Zeichen auf den Stack schreibt bzw. vom Stack löscht.*

Lösung:

(a) Der Kellerautomat $\mathcal{M}_1$ erkennt die Spiegelsprache $L = \{w\#w^R \mid w \in \Sigma^*\}$.

(b) Die durchlaufene Zustandsfolge ist $q_S, q_0, q_0, q_0, q_1, q^{\mathsf{1}}, q^{\mathsf{10}}, q_1', q_1, q^{\mathsf{0}}, q^{\mathsf{01}}, q_1', q_1, q^{\mathsf{0}}, q^{\mathsf{00}}$. Der Stackinhalt von $\mathcal{M}_2$, wenn er zum ersten Mal in den Zustand q_1 übergeht, ist $\mathsf{100100}$.

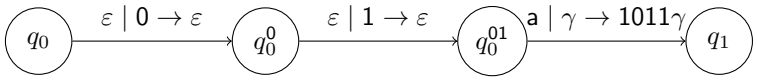
(c) Jeder 0-1-Kellerautomat ist auch ein Kellerautomat.

Umgekehrt sei $\mathcal{M}$ ein nichtdeterministischer Kellerautomat mit Stackalphabet Γ der Größe k. Wir konstruieren einen 0-1-Kellerautomaten $\mathcal{M}'$, der dieselbe Sprache wie $\mathcal{M}$ erkennt.

Die Symbole eines Stackalphabets der Länge k können als Binärstrings der Länge $\lceil \log_2(k) \rceil$ kodiert werden, indem wir beispielsweise die Symbole durchnummerieren und die binäre Repräsentation der Nummerierung verwenden. Der 0-1-Kellerautomat arbeitet prinzipiell gleich wie $\mathcal{M}$, jedes Symbol auf dem Stack von $\mathcal{M}$ ist allerdings durch $\lceil \log_2(k) \rceil$ Symbole auf dem Stack von $\mathcal{M}'$ kodiert. Die Übergangsfunktion von $\mathcal{M}'$ hängt also nicht nur vom obersten Symbol des Stacks ab, sondern von den obersten $\lceil \log_2(k) \rceil$ Symbolen. Wenn $\mathcal{M}$ das oberste Symbol liest, liest $\mathcal{M}'$ stattdessen in $\lceil \log_2(k) \rceil$ ε-Übergängen die obersten $\lceil \log_2(k) \rceil$ Symbole und speichert dabei die bereits gelesenen Symbole im aktuellen Zustand. Danach liest $\mathcal{M}'$ das aktuelle Symbol der Eingabe. Legt $\mathcal{M}$ ein Symbol auf den Stack, legt $\mathcal{M}'$ stattdessen die Kodierung des Symbols auf den Stack.

Am Anfang muss es einen ε-Übergang geben, bei dem die Kodierung des initialen Stacksymbols von $\mathcal{M}$ auf den Stack von $\mathcal{M}'$ gelegt wird.

Beispiel:
Aus einem Übergang $\delta(q_0, \mathsf{a}, A) = \delta(q_1, BC)$ in $\mathcal{M}$ wird in $\mathcal{M}'$ die folgende Übergangssequenz, wenn wir A mit $\mathsf{01}$, B mit $\mathsf{10}$ und C mit $\mathsf{11}$ kodieren:



Problem 6: Maschinenmodelle $3 + 2 + 4 = 9$ Punkte

Die Komplexitätsklasse $\mathsf{DTAPE}(x)$ ist definiert als die Menge der Sprachen, die von einer deterministischen Turingmaschine mit Platzbedarf höchstens x entschieden werden können. Dabei wird angenommen, dass die Turingmaschine über ein separates Arbeitsband (mit eigenem Kopf) verfügt, auf dem der Platzbedarf gemessen wird. Auf dem Eingabeband darf nicht geschrieben werden.

Wir betrachten in dieser Aufgabe die Klassen $\mathsf{DTAPE}(x)$, bei denen x eine Konstante ist, also nicht von der Eingabelänge abhängt. Der Einfachheit halber dürfen Sie davon ausgehen, dass die x Felder des Arbeitsbandes, auf denen geschrieben werden darf, entsprechend markiert sind. Die Turingmaschine kann also erkennen, wenn sie aus dem Bereich hinausläuft, in dem geschrieben werden darf.

Sei Σ ein beliebiges, aber festes Alphabet. Zu einem Wort $w \in \Sigma^*$ bezeichne w^R das gespiegelte Wort. Wir betrachten folgende Sprache:

$$L = \{uvu^R \mid u, v \in \Sigma^*, |u| = 2022\}$$

Diese Sprache enthält also alle Wörter, bei denen die ersten 2022 Zeichen gespiegelt den letzten 2022 Zeichen entsprechen, aber nicht mit diesen überlappen.

(a) Geben Sie eine deterministische Turingmaschine mit Platzbedarf 2022 an, die L entscheidet und höchstens 1 Million Zustände benutzt. Sie müssen nicht begründen, dass Ihre Turingmaschine höchstens 1 Million Zustände benutzt.

(b) Zeigen Sie: $L \in \mathsf{DTAPE}(0)$.

(c) Zeigen Sie: $\mathsf{DTAPE}(2022) = \mathsf{DTAPE}(0)$.

Lösung:

(a) Prüfe zunächst, dass die Eingabe mindestens 4044 Zeichen lang ist. Kopiere dazu zunächst die ersten 2022 Zeichen der Eingabe auf das Arbeitsband. Sobald das Arbeitsband voll ist, lösche es und kopiere die nächste 2022 Zeichen. Falls das Ende der Eingabe erreicht ist, bevor zum zweiten Mal das Ende des Arbeitsbandes erreicht wird, ist die Eingabe zu kurz und wird abgelehnt.

Ansonsten beginnt die zweite Phase, in der die ersten und letzten 2022 Zeichen verglichen werden: Kopiere die ersten 2022 Zeichen der Eingabe auf das Arbeitsband. Gehe dann auf dem Eingabeband zum Ende der Eingabe und auf dem Arbeitsband zum Anfang des kopierten Worts. Gehe nun auf dem Eingabeband von rechts nach links und auf dem Arbeitsband von links nach rechts und vergleiche die Wörter Zeichen für Zeichen. Wenn zwei Zeichen nicht übereinstimmen, lehne ab. Wenn alle Zeichen übereinstimmen, akzeptiere.

(b) Die Länge der Eingabe kann geprüft werden, indem über den Zustand der Turingmaschine bis 4044 gezählt wird. Um die Zeichen zu vergleichen, muss sich die Turingmaschine die ersten 2022 Zeichen im Zustand merken. Das ist möglich, weil es nur konstant viele Zeichen sind.

(c) $\mathsf{DTAPE}(0) \subseteq \mathsf{DTAPE}(2022)$ ist trivialerweise erfüllt.

$\mathsf{DTAPE}(2022) \subseteq \mathsf{DTAPE}(0)$: Sei $\mathcal{M} = (Q, \Sigma, \sqcup, \Gamma, s, \delta, F)$ eine Turingmaschine mit Platzbedarf höchstens 2022. Konstruiere eine Turingmaschine $\mathcal{M}'$ ohne Arbeitsband und $L(\mathcal{M}) = L(\mathcal{M}')$. Kodiere die 2022 Bandzeichen auf dem Arbeitsband von $\mathcal{M}$ im Zustand von $\mathcal{M}'$. Zusätzlich muss im Zustand kodiert werden, auf welchem der 2022 Zeichen der Arbeitskopf von $\mathcal{M}$ steht. Insgesamt ergibt das $2022 \cdot |\Gamma|^{2022}$ verschiedene Bandkonfigurationen. Für jede mögliche Bandkonfiguration und jeden Zustand $q \in Q$ hat $\mathcal{M}'$ eine Kopie von q. Beim Ausführen eines Übergangs muss $\mathcal{M}'$ ausgehend von der im Zustand gespeicherten Bandkonfiguration die nachfolgende Bandkonfiguration berechnen und in den entsprechenden Zustand übergehen.

Problem 7: Entscheidbarkeit $1 + 2 + 3 = 6$ Punkte

Sei $\Sigma = \{0, 1\}$ ein Alphabet. Aus der Vorlesung wissen Sie, dass das Halteproblem

$$\mathcal{H} = \{\langle \mathcal{M} \rangle \# w \mid \mathcal{M} \text{ hält bei Eingabe } w\}$$

unentscheidbar ist.

Betrachten Sie nun die folgende Sprache:

$$\mathcal{H}_{\text{eq}} = \{\langle \mathcal{M}_1 \rangle \# \langle \mathcal{M}_2 \rangle \mid \text{ für alle } v \in \Sigma^* \text{ gilt: } \mathcal{M}_1 \text{ hält auf } v \iff \mathcal{M}_2 \text{ hält auf } v\}$$

Im Folgenden sollen Sie zeigen, dass auch $\mathcal{H}_{\text{eq}}$ unentscheidbar ist.

(a) Konstruieren Sie eine Turingmaschine $\mathcal{N}$, sodass für jede Turingmaschine $\mathcal{M}$ gilt:

$$\langle \mathcal{N} \rangle \# \langle \mathcal{M} \rangle \in \mathcal{H}_{\text{eq}} \iff \mathcal{M} \text{ hält auf jeder Eingabe } v \in \Sigma^*$$

(b) Seien $\mathcal{M}$ eine Turingmaschine und $w \in \Sigma^*$ beliebig, aber fest. Konstruieren Sie eine Turingmaschine $\mathcal{N}_{w,\mathcal{M}}$ mit der folgenden Eigenschaft:

$$\mathcal{N}_{w,\mathcal{M}} \text{ hält auf jeder Eingabe } v \in \Sigma^* \iff \mathcal{M} \text{ hält auf } w$$

Ihre Konstruktion muss berechenbar sein.

(c) Zeigen Sie, dass $\mathcal{H}_{\text{eq}}$ nicht entscheidbar ist. Reduzieren Sie vom Halteproblem. Verwenden Sie nicht den Satz von Rice!

Lösung:

(a) Gilt für ein Wort $\langle \mathcal{N} \rangle \# \langle \mathcal{M} \rangle \in \mathcal{H}_{\text{eq}}$, haben beide Turingmaschinen das gleiche Halteverhalten. Die Turingmaschine $\mathcal{N}$, die jede Eingabe verwirft und sofort hält, hat also die gewünschten Eigenschaften.

(b) Seien $\mathcal{M}$ eine Turingmaschine und $w \in \Sigma^*$ gegeben. Sei $\mathcal{N}_{w,\mathcal{M}}$ eine Turingmaschine, die alle Eingaben verwirft, immer $\mathcal{M}$ auf w simuliert und das Akzeptanzverhalten von $\mathcal{M}$ übernimmt. Dann hat $\mathcal{N}_{w,\mathcal{M}}$ die gewünschten Eigenschaften.

(c) Wir nehmen an, dass $\mathcal{H}_{\text{eq}}$ von einer Turingmaschine $\mathcal{M}_{\text{eq}}$ entschieden wird und konstruieren daraus eine Turingmaschine $\mathcal{M}_{\mathcal{H}}$, die das Halteproblem entscheidet. Für eine Instanz $\langle \mathcal{M} \rangle \# w$ vom Halteproblem arbeitet $\mathcal{M}_{\mathcal{H}}$ wie folgt:

$\mathcal{M}_{\mathcal{H}}$ konstruiert eine Turingmaschine $\mathcal{N}$ wie in Teilaufgabe (a) und eine Turingmaschine $\mathcal{N}_{w,\mathcal{M}}$ wie in Teilaufgabe (b). Dann simuliert $\mathcal{M}_{\mathcal{H}}$ die Turingmaschine $\mathcal{M}_{\text{eq}}$ auf der Eingabe $\langle \mathcal{N} \rangle \# \langle \mathcal{N}_{w,\mathcal{M}} \rangle$. $\mathcal{M}_{\mathcal{H}}$ akzeptiert genau dann, wenn $\mathcal{M}_{\text{eq}}$ akzeptiert.

Aus den vorherigen Teilaufgaben folgt:

$$\langle \mathcal{N} \rangle \# \langle \mathcal{N}_{w,\mathcal{M}} \rangle \in \mathcal{H}_{\text{eq}} \overset{(a)}{\iff} \mathcal{N}_{w,\mathcal{M}} \text{ hält auf jeder Eingabe } v \in \Sigma^* \overset{(b)}{\iff} \mathcal{M} \text{ hält auf } w$$

Damit entscheidet $\mathcal{M}_{\mathcal{H}}$ das Halteproblem, was aber ein Widerspruch ist. Daher kann $\mathcal{M}_{\text{eq}}$ nicht entscheidbar sein.