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

Lösung!

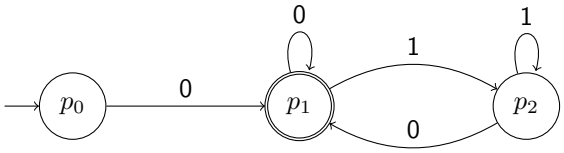
- Bringen Sie den Aufkleber mit Ihrem Namen und Ihrer Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihrem Namen und Matrikelnummer.
- Schreiben Sie die Lösungen auf die Aufgabenblätter und Rückseiten. Am Ende der Klausur sind zusätzliche Leerseiten. Fordern Sie zusätzliches 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	Σ	a	b	c	d	Σ
Aufg. 1	2	3	4	–	9				–	
Aufg. 2	1	3	2	4	10					
Aufg. 3	1	1	1	7	10					
Aufg. 4	2	3	2	–	7				–	
Aufg. 5	3	3	–	–	6			–	–	
Aufg. 6	2	5	3	–	10				–	
Aufg. 7	1	3	1	3	8					
Σ					60					

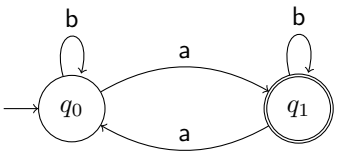
Problem 1: Endliche Automaten 2 + 3 + 4 = 9 Punkte

Sei $\mathcal{A}$ ein endlicher Automat über dem Alphabet $\Sigma = \{0, 1\}$ und sei $\mathcal{A}_{\text{ins}}$ ein endlicher Automat über dem Alphabet $\Sigma_{\text{ins}} = \{\mathsf{a}, \mathsf{b}\}$. Diese sind durch folgende Übergangsgraphen (mit implizitem Fehlerzustand) definiert:

$\mathcal{A}$:



$\mathcal{A}_{\text{ins}}$:



Sei L die Sprache, die von $\mathcal{A}$ erkannt wird, und L_{ins} die Sprache, die von $\mathcal{A}_{\text{ins}}$ erkannt wird.

(a) Geben Sie L und L_{ins} an.

Wir betrachten nun ein Wort $w \in L$ und fügen nach jeder 1 in w ein beliebiges Wort aus L_{ins} ein.

Beispiel: $w = 011010 \rightsquigarrow 01\mathsf{a}1\mathsf{a}01\mathsf{babbaa}0, 01\mathsf{abb}1\mathsf{a}01\mathsf{bab}0$

Die Sprache aller Wörter, die so entstehen können, bezeichnen wir mit $\alpha(w, 1, L_{\text{ins}})$. Außerdem definieren wir

$$\alpha(L, 1, L_{\text{ins}}) = \bigcup_{w \in L} \alpha(w, 1, L_{\text{ins}}).$$

Das heißt, $\alpha(L, 1, L_{\text{ins}})$ enthält alle Wörter, die entstehen, wenn wir mit einem beliebigen Wort aus L beginnen und nach jeder 1 ein beliebiges Wort aus L_{ins} einfügen.

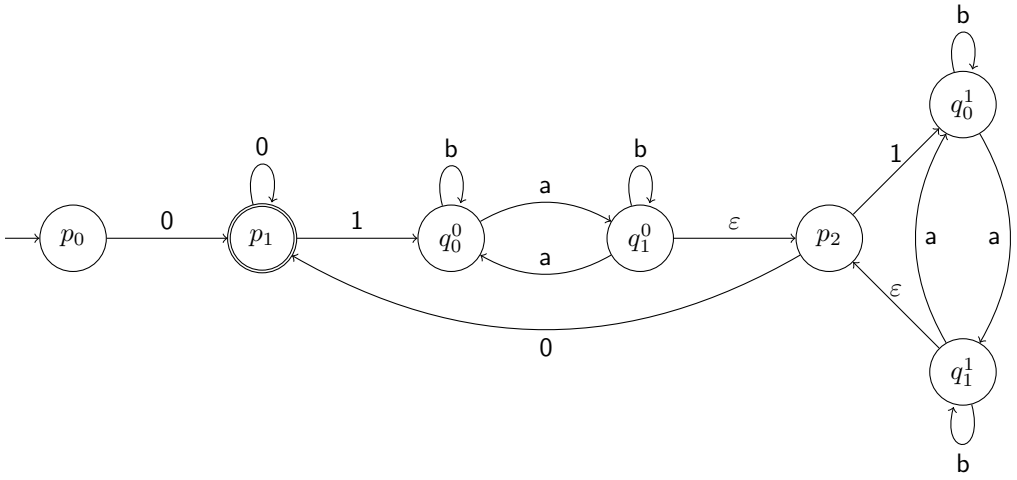
(b) Geben Sie einen nichtdeterministischen endlichen Automaten für $\alpha(L, 1, L_{\text{ins}})$ an, mit L und L_{ins} wie oben.

(c) Seien nun L und L_{ins} beliebige reguläre Sprachen über $\Sigma = \{0, 1\}$ bzw. $\Sigma_{\text{ins}} = \{\mathsf{a}, \mathsf{b}\}$. Zeigen Sie, dass $\alpha(L, 1, L_{\text{ins}})$ ebenfalls regulär ist. Zeigen Sie dazu, dass ein Automat $\mathcal{A}_\alpha$ existiert, der $\alpha(L, 1, L_{\text{ins}})$ erkennt.

Lösung:

(a) $L(\mathcal{A}) = \{w \in \Sigma^* \mid w \text{ beginnt mit } 0 \text{ und endet mit } 0\}$
$L(\mathcal{A}_{\text{ins}}) = \{w \in \Sigma_{\text{ins}}^* \mid \text{Anzahl der } \mathsf{a}\text{s in } w \text{ ist ungerade}\}$

(b) Der Automat für $\alpha(L, 1, L_{\text{ins}})$ sieht wie folgt aus:



(c) Da L und L_{ins} regulär sind, existieren auch zwei endliche Automaten $\mathcal{A}$ und $\mathcal{A}_{\text{ins}}$, die L bzw. L_{ins} erkennen. Daraus konstruieren wir einen neuen Automaten $\mathcal{A}_\alpha$ über dem Alphabet $\Sigma_\alpha = \{\mathsf{a}, \mathsf{b}, \mathsf{0}, \mathsf{1}\}$, der $\alpha(L, \mathsf{1}, L_{\text{ins}})$ erkennt. Die Idee ist, in $\mathcal{A}$ jeden 1-Übergang durch eine Kopie des Automaten $\mathcal{A}_{\text{ins}}$ zu erweitern. Dafür wird jeder 1-Übergang mit $\delta(q_1, 1) = q_2$ $(q_1, q_2 \in Q)$ in dem Automaten $\mathcal{A}$ wie folgt modifiziert:

- Der Übergang wird gelöscht.
- Eine Kopie von $\mathcal{A}_{\text{ins}}$ wird erstellt, wobei alle Zustände der Kopie als nicht akzeptierend definiert sind.
- Von q_1 kommt man mit einem 1-Übergang zum Startzustand der Kopie. Die Zustände, die in $\mathcal{A}_{\text{ins}}$ akzeptierend sind, sind jeweils durch ε-Übergänge mit q_2 verbunden.

Der Startzustand bzw. die akzeptierenden Zustände von $\mathcal{A}_\alpha$ entsprechen dem Startzustand bzw. den akzeptierenden Zuständen von $\mathcal{A}$.

Wir zeigen nun $L(\mathcal{A}_\alpha) = \alpha(L, \mathsf{1}, L_{\text{ins}})$.

$\supseteq$ Betrachte ein Wort w aus $\alpha(L, \mathsf{1}, L_{\text{ins}})$. Der Automat arbeitet wie $\mathcal{A}$, außer wenn er eine 1 liest. In diesem Fall geht er in eine Kopie von $\mathcal{A}_{\text{ins}}$ und fährt nach Verlassen der Kopie wie $\mathcal{A}$ fort, da das Subwort aus as und bs in L_{ins} ist und somit von $\mathcal{A}_{\text{ins}}$ akzeptiert wird. Da w ohne die as und bs in L liegt, wird es von $\mathcal{A}$ akzeptiert. Also wird auch w von $\mathcal{A}_\alpha$ akzeptiert.

$\subseteq$ Sei w ein Wort, das von $\mathcal{A}_\alpha$ akzeptiert wird. Da die Struktur von $\mathcal{A}_\alpha$ mit der von $\mathcal{A}$ übereinstimmt, ist w eingeschränkt auf die 0en und 1en in L. Zu zeigen ist also noch, dass jedes maximale Teilwort w' aus as und bs nach einer 1 kommt und in L_{ins} enthalten ist. Ersteres gilt nach Konstruktion. Letzteres gilt, da nach Verlassen einer Kopie von $\mathcal{A}_{\text{ins}}$ zuerst eine 1 gelesen werden muss, bevor eine neue Kopie betreten werden kann. Folglich wurde w' fertig gelesen, wenn die Kopie durch einen ε-Übergang von einem akzeptierenden Zustand von $\mathcal{A}_{\text{ins}}$ verlassen wird, also wird w' von $\mathcal{A}_{\text{ins}}$ akzeptiert.

Problem 2: Grammatiken/Kodierungen $1 + 3 + 2 + 4 = 10$ Punkte

In dieser Aufgabe zeigen wir, dass neben dem Startsymbol zwei Variablen ausreichen, um beliebige Grammatiken zu konstruieren. Wir betrachten zunächst folgende Beispielgrammatik.

Beispiel: $G_{\text{Bsp}} = (\Sigma_{\text{Bsp}}, V_{\text{Bsp}}, S_{\text{Bsp}}, R_{\text{Bsp}})$ mit $\Sigma_{\text{Bsp}} = \{2, 3, 4\}$, $V_{\text{Bsp}} = \{S_{\text{Bsp}}, X_1, X_2, X_3, X_4\}$ und

$$\begin{aligned} R_{\text{Bsp}} = \{ \quad & S_{\text{Bsp}} \to X_2 \\ & X_1 \to X_2 \\ & X_2 \to 2 \\ & X_3 \to 3 \\ & X_4 \to 4 \\ & X_1X_1 \to X_1X_1X_2X_3 \\ \}. & \end{aligned}$$

(a) Welche Sprache erzeugt G_{Bsp}?

Sei Σ ein Alphabet und $G = (\Sigma, V, S, R)$ eine Grammatik mit der Variablenmenge $V = \{S, X_1, \ldots, X_n\}$ für ein $n \in \mathbb{N}$. Für eine Funktion $f\colon \{X_1, \ldots, X_n\} \to \{A, B\}^*$ definieren wir die *f-Kodierung von G* als die Grammatik $G' = (\Sigma, V', S, R')$ mit $V' = \{S, A, B\}$ und der Regelmenge R', die aus R entsteht, indem jedes X_i für $i \in \{1, \ldots, n\}$ durch $f(X_i)$ ersetzt wird.

Beispiel: Sei f gegeben durch $f(X_1) = ABA$, $f(X_2) = ABBA$, $f(X_3) = ABBBA$ und $f(X_4) = ABBBBA$. Dann ergibt sich die folgende Regelmenge R' für die f-Kodierung G'_{Bsp} von G_{Bsp}:

$$\begin{aligned} R' = \{ \quad & S_{\text{Bsp}} \to ABBA \\ & ABA \to ABBA \\ & ABBA \to 2 \\ & ABBBA \to 3 \\ & ABBBBA \to 4 \\ & ABA\,ABA \to ABA\,ABA\,ABBA\,ABBBA \\ \}. & \end{aligned}$$

Dr. Meta möchte die Variablen möglichst effizient kodieren und schlägt vor, eine Huffman-Kodierung zu verwenden. Für eine Grammatik $G = (\Sigma, V, S, R)$ mit $V = \{S, X_1, \ldots, X_n\}$ definieren wir die Huffman-Kodierung von G wie folgt. Für ein X_i mit $i \in \{1, \ldots, n\}$ ist die Wahrscheinlichkeit p_i definiert als die relative Häufigkeit von X_i in R, das heißt

$$p_i = \frac{\text{Anzahl Vorkommen von } X_i \text{ in } R}{\sum_{j=1}^{n} \text{Anzahl Vorkommen von } X_j \text{ in } R}$$

Dabei werden die Vorkommen auf *beiden* Seiten der Regeln gezählt.

Beispiel: Die Anzahl Vorkommen von X_1 in R ist 5, d.h. $p_1 = 5/12$.

Sei f eine Huffman-Kodierung für $\{X_1, \ldots, X_n\}$ mit den so definierten Wahrscheinlichkeiten $p_1, \ldots, p_n$. Die f-Kodierung von G heißt dann auch *Huffman-Kodierung von G*. Der zu f gehörende Huffman-Baum heißt *Huffman-Baum der Grammatik G*.

(b) Geben Sie einen Huffman-Baum für die Grammatik G_{Bsp} an. Sortieren Sie dafür die Variablen von links nach rechts nach absteigender Wahrscheinlichkeit und wählen Sie A für den linken Ast und B für den rechten Ast. Geben Sie die Kodierung zu Ihrem Huffman-Baum durch Wörter aus $\{A, B\}^*$ in folgender Tabelle an.

Variable	Kodierung
X_1	A
X_2	BA
X_3	BBA
X_4	BBB

(c) Zeigen Sie, dass Dr. Metas Ansatz im Allgemeinen nicht funktioniert. Zeigen Sie hierfür, dass die Grammatik G'_{Bsp}, die durch die Huffman-Kodierung entsteht, nicht dieselbe Sprache wie G_{Bsp} erzeugt.

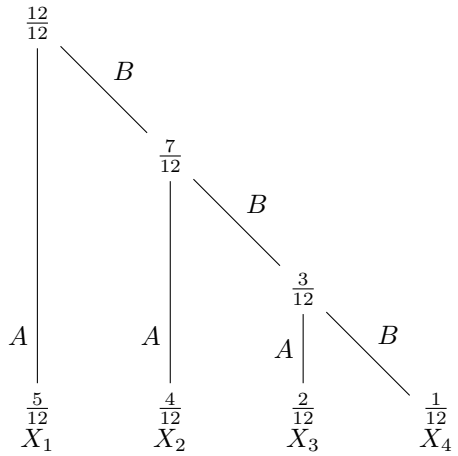
(d) Wir möchten dennoch zeigen, dass neben dem Startsymbol zwei Variablen ausreichen. Sei $G = (\Sigma, V, S, R)$ eine beliebige Grammatik mit $V = \{S, X_1, \ldots, X_n\}$ für ein $n \in \mathbb{N}$. Zeigen Sie, dass eine äquivalente Grammatik $G' = \{\Sigma, V', S, R'\}$ mit $V' = \{S, A, B\}$ existiert.

Hinweis: Verwenden Sie eine Unär-Kodierung mit Trennzeichen, das heißt $f(X_i) = AB^iA$ für $i \in \{1, \ldots, n\}$, siehe Beispiel.

Lösung:

(a) $L(G_{\text{Bsp}}) = \{2\}$

(b) Der Huffman-Baum sieht wie folgt aus, Kodierung siehe oben.



(c) G_{Bsp} erzeugt die Sprache $L(G_{\text{Bsp}}) = \{2\}$. Insbesondere ist das Wort 3 nicht in $L(G_{\text{Bsp}})$ enthalten, da X_3 nie auf der rechten Seite steht.

Das Wort 3 kann mit R' jedoch wie folgt abgeleitet werden: $S_{\text{Bsp}} \to BA \to BBA \to 3$. Also ist $3 \in L(G'_{\text{Bsp}})$ ein Zeuge dafür, dass $L(G_{\text{Bsp}}) \neq L(G'_{\text{Bsp}})$.

(d) Wir wählen $f(X_i) = AB^iA$ für $i \in \{1, \ldots, n\}$ und G' als die f-Kodierung von G. Die Regelmenge R' entsteht also aus R, indem für $i = 1, \ldots, n$ jedes X_i durch AB^iA ersetzt wird. Wir nennen AB^iA das *Codewort* von X_i. Damit verwendet R' nur Variablen aus V'.

Wir zeigen nun $L(G) = L(G')$.

- $\subseteq$ Für ein Wort $w \in L(G)$ betrachten wir eine Folge $S \xrightarrow{*}_G w$ von Ableitungen aus R. Ersetzen von X_i durch AB^iA für jedes $i = 1, \ldots, n$ ergibt eine Folge $S \xrightarrow{*}_{G'} w$ von Ableitungen aus R', also ist $w \in L(G')$.
- $\supseteq$ Sei nun $w \in L(G')$ und $S \xrightarrow{*}_{G'} w$ eine Folge von Ableitungen aus R'. Betrachte eine Ableitung $\alpha \to \beta$ mit $\alpha, \beta \in \{AB^iA \mid 1 \leqslant i \leqslant n\}^*$ aus R'. Nach Konstruktion beginnen alle Codewörter sowie α mit AB und enden mit BA. Die Folgen AB und BA kommen außerdem nur zu Beginn bzw. am Ende eines Codewortes vor. Ein Codewort kann also nicht als Teilwort in einer Folge von anderen Codewörtern vorkommen. Also ersetzt $\alpha \to \beta$ jedes Codewort entweder vollständig oder gar nicht. In beiden Fällen kann in $S \xrightarrow{*}_{G'} w$ jedes Codewort AB^iA durch X_i ersetzt werden, was eine Folge $S \xrightarrow{*}_G w$ von Ableitungen aus R ergibt.

Anmerkung: Es reicht nicht, zu zeigen, dass kein Codewort ein Teilwort eines anderen Codeworts ist.

Betrachte dazu die Grammatik $G = (\Sigma, V, S, R)$ mit $\Sigma = \{1, 2, 4\}$, $V = \{S, X_1, X_2, X_3, X_4\}$ und

$$\begin{aligned} R = \{ \quad & S \to X_2X_2 \\ & X_1 \to 1 \\ & X_2 \to 2 \\ & X_3 \to X_2 \\ & X_4 \to 4 \\ \}. & \end{aligned}$$

Diese Grammatik erzeugt die Sprache $L(G) = \{22\}$.

Betrachte die Kodierung f mit $f(X_1) = AA$, $f(X_2) = AB$, $f(X_3) = BA$ und $f(X_4) = BB$. Diese Kodierung erfüllt die Eigenschaft, dass kein Codewort ein Teilwort eines anderen Codeworts ist. Dennoch ändert sich durch die Kodierung mit f die erzeugte Sprache. Es ergibt sich folgende Regelmenge:

$$\begin{aligned} R' = \{ \quad & S \to ABAB \\ & AA \to 1 \\ & AB \to 2 \\ & BA \to AB \\ & BB \to 4 \\ \}. & \end{aligned}$$

In der kodierten Grammatik ist somit die Ableitungsfolge $S \to ABAB \to AABB \xrightarrow{*} 14 \notin L(G)$ möglich.

Problem 3: NP-Vollständigkeit $1 + 1 + 1 + 7 = 10$ Punkte

Aus der Vorlesung kennen Sie das folgende NP-vollständige Problem PARTITION:

PARTITION	
Gegeben:	Eine endliche Menge M, eine Gewichtsfunktion $w\colon M \to \mathbb{N}_0$
Frage:	Existieren disjunkte Teilmengen $M_1, M_2 \subseteq M$ mit $M_1 \cup M_2 = M$ und $w(M_1) = w(M_2)$?

Für eine Menge M' definieren wir das Gewicht von M' als die Summe der Gewichte der Elemente von M', d.h. $w(M') \coloneqq \sum_{m \in M'} w(m)$.

Wir betrachten nun das folgende Problem PARTITION-IN-3:

PARTITION-IN-3	
Gegeben:	Eine endliche Menge M, eine Gewichtsfunktion $w\colon M \to \mathbb{N}_0$
Frage:	Existieren paarweise disjunkte Teilmengen $M_1, M_2, M_3 \subseteq M$ mit $M_1 \cup M_2 \cup M_3 = M$ und $w(M_1) = w(M_2) = w(M_3)$?

(a) Geben Sie für folgende Instanz von PARTITION eine Lösung an.

$$\begin{aligned} M &= \{a, b, c, d, e, f\} \\ w(a) &= 0 \\ w(b) &= 2 \\ w(c) &= 3 \\ w(d) &= 4 \\ w(e) &= 5 \\ w(f) &= 6 \end{aligned}$$

(b) Sei (M, w) die PARTITION-Instanz aus (a). Konstruieren Sie eine Ja-Instanz (M', w') für PARTITION-IN-3 mit $M' = M \cup \{g\}$ und $w'(m) = w(m)$ für alle $m \in M$. Es genügt, wenn Sie das Gewicht von g angeben.

(c) Zeigen Sie, dass PARTITION-IN-3 in NP liegt. Geben Sie an, woraus ein Lösungsvorschlag für eine PARTITION-IN-3-Instanz besteht. Geben Sie außerdem die Laufzeit der Überprüfung im $\mathcal{O}$-Kalkül an und begründen Sie diese.

(d) Zeigen Sie, dass PARTITION-IN-3 NP-schwer ist.

Lösung:

(a) $M_1 = \{b, c, e\}, M_2 = \{a, d, f\}$

(b) $M' = M \cup \{g\}, w'(m) = \begin{cases} w(m) & \text{falls } m \in M \\ 10 & \text{falls } m = g \end{cases}$

(c) Eine Lösung von PARTITION-IN-3 besteht aus einer Zuordnung der Elemente von M zu je genau einer von drei Teilmengen, d.h. aus einer Funktion $f\colon M \to \{1,2,3\}$. Um die Korrektheit zu überprüfen, summieren wir für jede Teilmenge die Gewichte der Elemente auf und prüfen danach, ob die drei Ergebnisse übereinstimmen. Falls ja, akzeptieren wir, sonst lehnen wir ab. Dabei werten wir die Gewichtsfunktion für jedes Element einmal aus, summieren linear viele Elemente und führen eine konstante Anzahl Vergleiche aus. Insgesamt brauchen wir also $\mathcal{O}(|M|)$ Zeit.

Anmerkung: Statt einer Funktion können auch drei Mengen als Lösung verwendet werden. In diesem Fall muss überprüft werden, ob die drei Mengen eine Partition von M sind.

(d) Wir zeigen nun die NP-Schwere durch eine polynomielle Reduktion von PARTITION zu PARTITION-IN-3. Sei (M, w) eine PARTITION-Instanz. Wir definieren eine PARTITION-IN-3-Instanz (M', w') wie folgt. Falls $w(M)$ ungerade ist, dann setzen wir $M' = \{m'\}$ und $w'(m') = 1$. Falls $w(M)$ gerade ist, dann setzen wir

$$M' = M \cup \{m'\}, \text{ wobei } m' \notin M$$
$$w'(m) = \begin{cases} w(m) & \text{falls } m \in M \\ w(M)/2 & \text{falls } m = m'. \end{cases}$$

Hierfür werten wir w linear oft aus und summieren linear viele Werte. Die Reduktion ist also linear in der Eingabegröße.

Wir zeigen nun, dass (M, w) genau dann eine Ja-Instanz von PARTITION ist, wenn (M', w') eine Ja-Instanz von PARTITION-IN-3 ist.

$\Longrightarrow$ Sei (M, w) eine Ja-Instanz von PARTITION und (M_1, M_2) eine Lösung, insbesondere gilt $w(M_1) = w(M_2) = w(M)/2$. Also ist $w(M)/2 \in \mathbb{N}$, d.h. $w(M)$ ist gerade. Dann ist $(M_1, M_2, \{m'\})$ eine Lösung von PARTITION, denn $w'(m') = w(M)/2 = w(M_1) = w(M_2)$, $M_1 \cup M_2 \cup \{m'\} = M \cup \{m'\} = M'$ und die drei Mengen sind nach Voraussetzung (M_1 und M_2) bzw. nach Konstruktion (m') paarweise disjunkt.

$\Longleftarrow$ Sei (M', w') eine Ja-Instanz. Da eine einelementige Menge mit positivem Gewicht sich nicht in drei gleich schwere Mengen partitionieren lässt, gilt $M' = M \cup \{m'\}$ und $w'(m) = w(m)$ für $m \in M$ bzw. $w'(m') = w(M)/2$. Sei (M_1, M_2, M_3) eine Lösung für (M', w'), also $w'(M_1) = w'(M_2) = w'(M_3)$. Ohne Beschränkung der Allgemeinheit sei $m' \in M_1$. Nach Konstruktion ist $w'(M') = w'(M) + w'(m') = w(M) + w(M)/2 = 3/2w(M)$. Es folgt $w'(M_1) = w'(M_2) = w'(M_3) = w(M)/2$. Da m' schon Gewicht $w(M)/2$ hat und alle Gewichte nichtnegativ sind, enthält M_3 neben m' nur noch Elemente mit Gewicht 0. Sei nun $M_2' \coloneqq M_2 \cup M_3 \setminus \{m'\}$.

Wir zeigen, dass (M_1, M_2') eine Lösung von (M, w) ist.

- Da $M_1 \cap M_2 = \emptyset$ und $M_1 \cap M_3 = \emptyset$, folgt $M_1 \cap (M_2 \cup M_3) = \emptyset$, insbesondere also auch $M_1 \cap M_2' = \emptyset$.
- $M_1 \cup M_2' = M_1 \cup M_2 \cup M_3 \setminus \{m'\} = M' \setminus \{m'\} = M$
- Es gilt

$$\begin{aligned} w(M_2') &= \sum_{m \in M_2'} w(m) = \sum_{m \in M_2} w(m) + \sum_{m \in M_3 \setminus \{m'\}} w(m) \\ &= \sum_{m \in M_2} w(m) + \sum_{m \in M_3 \setminus \{m'\}} 0 \\ &= w(M_2) = w'(M_2) = w'(M_1) = w(M_1). \end{aligned}$$

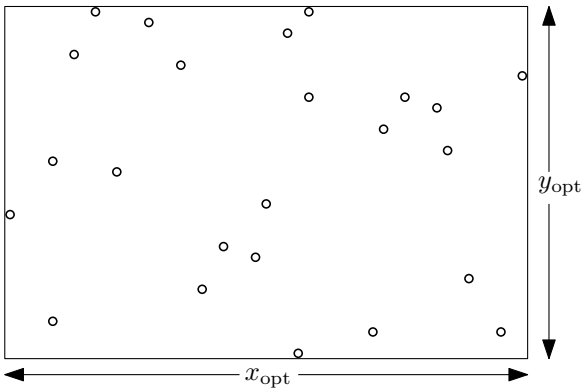
In der letzten Zeile geht ein, dass (M_1, M_2, M_3) eine Lösung von (M', w') ist und dass $m' \notin M_1, M_2$.

Problem 4: Approximation $2 + 3 + 2 = 7$ Punkte

Beim Minimierungsproblem SMALLESTBOUNDINGRECTANGLE ist eine Menge $P \subset \mathbb{R}^2$ von Punkten in der Ebene gegeben. Gesucht ist ein achsenparalleles[1] Rechteck mit minimalem Flächeninhalt, das alle Punkte in P enthält. Wir bezeichnen die Seitenlängen einer optimalen Lösung mit x_{opt} und y_{opt}.

[1]D.h. die Seiten des Rechtecks sind parallel zur x- bzw. y-Achse.

Beispiel:



(a) Zeigen Sie, dass SMALLESTBOUNDINGRECTANGLE optimal in Linearzeit gelöst werden kann.

Lösung:
Bestimme die Randkoordinaten des Rechtecks:

$$x_{\min} = \min_{(x,y)\in P} x$$
$$x_{\max} = \max_{(x,y)\in P} x$$
$$y_{\min} = \min_{(x,y)\in P} y$$
$$y_{\max} = \max_{(x,y)\in P} y$$

Das geht offensichtlich in Linearzeit. Das Rechteck wird dann durch die Eckpunkte $(x_{\min}, y_{\min})$, $(x_{\min}, y_{\max})$, $(x_{\max}, y_{\min})$ und $(x_{\max}, y_{\max})$ aufgespannt. Nach Konstruktion enthält das Rechteck alle Punkte in P. Wenn man $x_{\min}$ oder $y_{\min}$ vergrößert bzw. $x_{\max}$ oder $y_{\max}$ verkleinert, gibt es immer einen Punkt in P, der nicht mehr im Rechteck enthalten ist. Also sind beide Seitenlängen und damit der Flächeninhalt minimal.

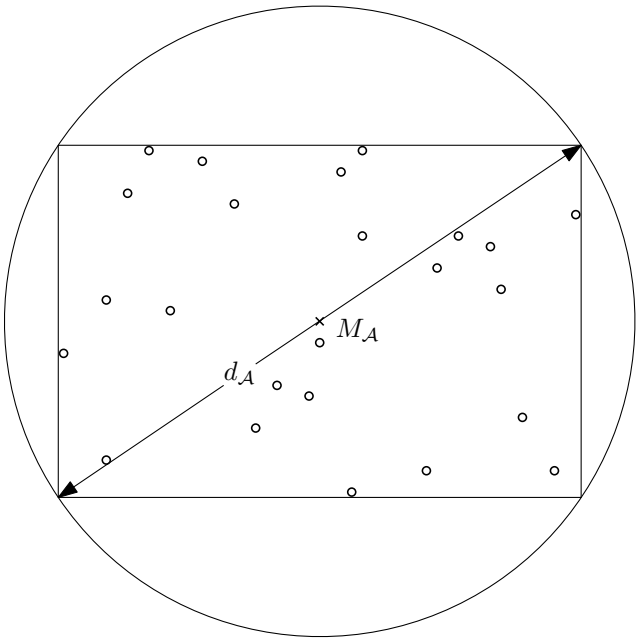
Wir betrachten nun das Minimierungsproblem SMALLESTBOUNDINGCIRCLE. Wieder ist eine Menge $P \subset \mathbb{R}^2$ von Punkten in der Ebene gegeben. Gesucht ist ein Kreis mit minimalem Durchmesser d_{opt}, der alle Punkte in P enthält.

(b) Zeigen Sie: $d_{\text{opt}} \geqslant \max(x_{\text{opt}}, y_{\text{opt}})$.

Lösung:
Für jedes Paar von Punkten in P muss der Durchmesser des Kreises mindestens so groß sein wie der Abstand zwischen den beiden Punkten. Sei o.B.d.A. x_{opt} die längere Seite des Rechtecks. Betrachte die zwei Punkte $(x_{\min}, y_1)$ und $(x_{\max}, y_2)$ mit minimaler bzw. maximaler x-Koordinate. Diese beiden Punkte haben Abstand $\geqslant x_{\text{opt}} = x_{\max} - x_{\min}$.

Wir betrachten nun folgenden Algorithmus $\mathcal{A}$ für SMALLESTBOUNDINGCIRCLE: Berechne zunächst eine optimale Lösung für SMALLESTBOUNDINGRECTANGLE. Seien $M_{\mathcal{A}}$ und $d_{\mathcal{A}}$ der Mittelpunkt und die Diagonallänge des berechneten Rechtecks. Erzeuge dann einen Kreis mit Mittelpunkt $M_{\mathcal{A}}$ und Durchmesser $d_{\mathcal{A}}$. Sie können davon ausgehen, dass es möglich ist, in Polynomialzeit Wurzeln zu ziehen.

Beispiel:



(c) Zeigen Sie, dass $\mathcal{A}$ ein polynomieller Approximationsalgorithmus für SMALLESTBOUNDINGCIRCLE mit relativer Gütegarantie $\sqrt{2}$ ist.

Lösung:
Die polynomielle Laufzeit folgt aus Aufgabenteil (a). Die Gültigkeit der Lösung folgt daraus, dass das Rechteck vollständig im Kreis enthalten ist und selbst alle Punkte in P enthält. Für den Durchmesser des Kreises gilt:

$$d_{\mathcal{A}} = \sqrt{x_{\text{opt}}^2 + y_{\text{opt}}^2} \leqslant \sqrt{2\max(x_{\text{opt}}, y_{\text{opt}})^2} = \sqrt{2} \cdot \max(x_{\text{opt}}, y_{\text{opt}}) \leqslant \sqrt{2} \cdot d_{\text{opt}}$$

Problem 5: Kellerautomaten 3 + 3 = 6 Punkte

(a) Seien Σ ein beliebiges Alphabet, $a, b \in \Sigma$ zwei beliebige Symbole aus dem Alphabet, und $k \in \mathbb{N}$ eine Konstante. Beschreiben Sie, wie ein deterministischer Kellerautomat die Sprache $L(a, b, k) = \{a^n b^{kn} \mid n \in \mathbb{N}\}$ erkennen kann.

Wir betrachten nun deterministische Kellerautomaten, die ihren Lesekopf frei auf dem Eingabeband bewegen können. Wir sagen, dass die Ausführung eines solchen Kellerautomaten beendet ist, sobald er das Feld direkt hinter dem letzten Zeichen der Eingabe erreicht. Das Akzeptanzverhalten ist dann wie üblich über einen leeren Stack oder einen akzeptierenden Endzustand definiert.

(b) Beschreiben Sie, wie ein solcher Kellerautomat die Sprache $L = \{\mathtt{0}^n\mathtt{1}^{2n}\mathtt{2}^{6n} \mid n \in \mathbb{N}\}$ erkennen kann.

Hinweis: Sie dürfen den Kellerautomaten aus Teilaufgabe (a) als Subroutine verwenden, auch wenn Sie Teilaufgabe (a) nicht gelöst haben. Falls Sie den Kellerautomaten anpassen müssen, gehen Sie auf Unterschiede ein.

Lösung:

(a) Gehe die Eingabe von links nach rechts durch. Solange a gelesen wird, lege für jedes gelesene a k-mal a auf den Stack. Sobald ein anderes Zeichen als a gelesen wird, entferne für jedes gelesene b ein a vom Stack. Wird irgendein anderes Symbol als b gelesen, lehne ab. Wenn die Eingabe die korrekte Form hat, erreicht der Automat genau dann das Ende des Wortes, wenn der Stack leer wird. Akzeptiere in diesem Fall, ansonsten lehne ab.

(b) Der Automat arbeitet in zwei Phasen. Zunächst wird geprüft, dass das Wort mit $\mathtt{0}^n\mathtt{1}^{2n}$ für ein $n \in \mathbb{N}$ beginnt. Dazu kann das Verfahren für $L(\mathtt{0}, \mathtt{1}, \mathtt{2})$ verwendet werden. Allerdings wird statt dem Ende des Wortes nach dem ersten Vorkommen von `2` gesucht. Falls die Überprüfung erfolgreich war, gehe zurück zur ersten `1`. Überprüfe nun, ob das restliche Wort die Form $\mathtt{1}^m\mathtt{2}^{3m}$ für $m \in \mathbb{N}$ hat. Verwende dazu das Verfahren für $L(\mathtt{1}, \mathtt{2}, 3)$.

Problem 6: Falltür-Turingmaschine $2 + 5 + 3 = 10$ Punkte

(a) Beschreiben Sie, wie die Sprache $L = \{1^k0^m1^m0^n \mid k, m, n \in \mathbb{N}_0\}$ von einer deterministischen Turingmaschine entschieden werden kann. Schreiben Sie dafür nur auf Feldern der Eingabe.

Sei $F \subseteq \Sigma^*$ eine Sprache über einem Alphabet Σ. Eine *F-Falltür-Turingmaschine* (F-FTTM) ist eine deterministische Turingmaschine, die zusätzlich über ein *Falltür-Modul* verfügt. Nach jeder Anwendung der Überführungsfunktion wird das Falltür-Modul aktiv und macht folgende Überprüfung: Falls die F-FTTM im letzten Schritt das Feld verändert hat (d.h. ein anderes Zeichen geschrieben hat als gelesen wurde), wird überprüft, ob das Wort auf dem Band in F ist. Falls ja, so wird die „Falltür“ ausgelöst, d.h. der gesamte Bandinhalt wird gelöscht.

Bei der Überprüfung werden alle Zeichen auf dem Band, die nicht in Σ enthalten sind, ignoriert. Das heißt, wenn $\Sigma = \{0, 1\}$ und $00 \in F$, so wird das Band auch dann gelöscht, wenn $0x0$ mit $x \notin \Sigma$ auf dem Band steht. Das Falltür-Modul arbeitet separat von der endlichen Kontrolleinheit, d.h. Zustand und Kopfposition werden durch das Falltür-Modul nicht verändert. Falls der Bandinhalt nicht gelöscht wird, so wird er durch das Falltür-Modul nicht verändert.

(b) Sei $\Sigma = \{0, 1\}$ und $F = \{w \in \Sigma^* \mid w = w^R\}$ die Palindromsprache. Beschreiben Sie, wie eine F-Falltür-Turingmaschine die Sprache $L = \{1^k0^m1^m0^n \mid k, m, n \in \mathbb{N}_0\}$ entscheidet.

Hinweis: Verhindern Sie, dass die Falltür ausgelöst wird, indem Sie die Felder vor und nach der Eingabe nutzen. Sie dürfen die Turingmaschine aus Aufgabenteil (a) als Subroutine nutzen, auch wenn Sie (a) nicht bearbeitet haben. Falls Sie die Turingmaschine anpassen müssen, gehen Sie auf Unterschiede ein.

(c) Zeigen Sie, dass es Sprachen F und L gibt, sodass L von einer F-FTTM entschieden werden kann, aber von keiner klassischen Turingmaschine.

Lösung:

(a) Wir konstruieren eine Turingmaschine $\mathcal{M}$, die L entscheidet. Ist die Eingabe leer, so akzeptiert $\mathcal{M}$. Andernfalls läuft $\mathcal{M}$ nach rechts bis zur ersten 0 und löscht dabei alle gelesenen Einsen. Falls das Eingabewort zu Ende ist, ohne dass $\mathcal{M}$ eine 0 liest, so wird akzeptiert. Andernfalls läuft $\mathcal{M}$ zum Ende des Wortes und löscht von dort aus alle Nullen bis zur rechtesten 1. Falls das Band leer ist, ohne dass $\mathcal{M}$ eine 1 liest, so wird akzeptiert.

Andernfalls steht $\mathcal{M}$ jetzt auf der letzten 1 und wiederholt folgende Schritte: $\mathcal{M}$ löscht die 1, läuft nach links bis zum Wortanfang und löscht die linkeste 0. Existiert diese nicht, so lehnt $\mathcal{M}$ ab. $\mathcal{M}$ läuft wieder nach rechts auf das letzte Zeichen auf dem Band. Falls $\mathcal{M}$ dabei feststellt, dass das Band leer ist, dann wird akzeptiert. Falls das letzte Zeichen eine 0 ist, dann lehnt $\mathcal{M}$ ab.

(b) Wir konstruieren eine F-Falltür-Turingmaschine $\mathcal{M}$, die L entscheidet. $\mathcal{M}$ startet auf dem ersten Feld der Eingabe. Falls dort eine 0 steht, so geht $\mathcal{M}$ in einen Zustand q_0. Falls dort eine 1 steht, so geht $\mathcal{M}$ in einen anderen Zustand q_1. Ist die Eingabe leer, so akzeptiert $\mathcal{M}$. Wir nehmen nun an, dass $\mathcal{M}$ eine 0 oder eine 1 gelesen hat und sich danach in Zustand q_i befindet. $\mathcal{M}$ läuft bis zum ersten Blank nach der Eingabe, geht von dort ein weiteres Feld nach rechts und schreibt $1 - i$. Damit sind auf dem Band das erste und letzte Zeichen aus Σ unterschiedlich und die Falltür bleibt geschlossen. Anschließend geht $\mathcal{M}$ ein Feld nach links auf das Blanksymbol und eine schreibt ein Trennzeichen #. Wir wiederholen das Vorgehen links der Eingabe, d.h. $\mathcal{M}$ läuft nach links bis zum ersten Blanksymbol, schreibt dort #, läuft ein Feld nach links und schreibt dort i. Das erste und das letzte Zeichen aus Σ unterscheiden sich wieder, sodass die Falltür geschlossen bleibt.

Um zu entscheiden, ob die Eingabe w ein Wort aus L' ist, arbeitet $\mathcal{M}$ nur noch mit den Feldern zwischen den Trennzeichen. Auf dem Band steht also nie ein Palindrom. $\mathcal{M}$ steht aktuell links des linken Trennzeichens und läuft von dort zwei Schritte nach rechts auf das erste Feld des Eingabewortes.

Von dort startend arbeitet $\mathcal{M}$ wie die Turingmaschine aus (a). Hierbei wird das Trennzeichen # wie ein Blank behandelt und markiert Anfang bzw. Ende des Wortes.

(c) Seien F und L beide das Halteproblem, also

$$F = L = \{\langle\mathcal{M}\rangle\#w \mid \langle\mathcal{M}\rangle \text{ ist Gödelnummer einer Turingmaschine } \mathcal{M}, w \in \{0,1\}^*\} \subseteq \Sigma^*$$

mit $\Sigma = \{0, 1, \#\}$. Aus der Vorlesung wissen wir, dass das Halteproblem für klassische Turingmaschinen unentscheidbar ist.

Wir konstruieren nun eine F-FTTM, die das Halteproblem entscheidet. Dazu wollen wir herausfinden, ob die Falltür auf der Eingabe auslöst. Falls die Eingabe leer ist, lehne ab. Andernfalls gehe vom ersten Symbol der Eingabe einen Schritt nach links auf ein Blanksymbol und schreibe dort ein Zeichen, das nicht in Σ ist. Dadurch wird die Überprüfung der Falltür ausgelöst. Gehe wieder einen Schritt nach rechts. Lesen wir nun ein Blanksymbol, so wurde die Falltür ausgelöst, d.h. die Eingabe ist in F, also auch in L und wir akzeptieren. Lesen wir ein Zeichen aus Σ, so wissen wir, dass die Falltür nicht ausgelöst wurde. Also ist die Eingabe nicht in F, also auch nicht in L und wir lehnen ab.

Problem 7: Entscheidbarkeit $1 + 3 + 1 + 3 = 8$ Punkte

Sei Σ ein beliebiges, aber festes Alphabet. Betrachten Sie das folgende Entscheidungsproblem Π: Gegeben kontextfreie Grammatiken G_1 und G_2 über Σ, ist der Schnitt $L(G_1) \cap L(G_2)$ leer? In der Vorlesung haben Sie gesehen, dass dieses Problem unentscheidbar ist.

(a) Beschreiben Sie eine Turingmaschine $\mathcal{M}'$, die als Eingabe eine kontextfreie Grammatik G und ein Wort $w \in \Sigma^*$ bekommt und entscheidet, ob $w \in L(G)$ gilt.

(b) Zeigen Sie, dass das Komplement von Π semi-entscheidbar ist.

(c) Ist Π auch semi-entscheidbar? Beweisen Sie.

(d) Seien nun statt kontextfreien Grammatiken zwei Turingmaschinen gegeben. Wir erhalten das neue Entscheidungsproblem Π': Gegeben zwei Turingmaschinen $\mathcal{M}_1$ und $\mathcal{M}_2$ mit Eingabealphabet Σ, ist der Schnitt von $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$ leer? Zeigen Sie, dass das Komplement von Π' ebenfalls semi-entscheidbar ist.

Hinweis: Beachten Sie, dass die Turingmaschinen $\mathcal{M}_1$ bzw. $\mathcal{M}_2$ die Sprachen $L(\mathcal{M}_1)$ bzw. $L(\mathcal{M}_2)$ nicht entscheiden müssen.

Lösung:

(a) Die Turingmaschine $\mathcal{M}'$ transformiert die Grammatik G in Chomsky-Normalform und wendet dann den CYK-Algorithmus an.

(b) Wir wollen eine Turingmaschine $\mathcal{M}$ konstruieren, die zwei kontextfreie Grammatiken G_1, G_2 genau dann akzeptiert, wenn $L(G_1) \cap L(G_2) \neq \emptyset$ gilt.

Für zwei gegebene kontextfreie Grammatiken G_1 und G_2 zählt $\mathcal{M}$ Wörter in Σ^* in kanonischer Reihenfolge auf. Für jedes Wort $w \in \Sigma^*$ simuliert $\mathcal{M}$ dann die Turingmaschine $\mathcal{M}'$ aus (a) auf G_1 und w bzw. G_2 und w. Wenn $\mathcal{M}'$ für beide Grammatiken akzeptiert, ist $w \in L(G_1) \cap L(G_2)$ und $\mathcal{M}$ hält und akzeptiert. Sonst wird das nächste Wort aufgezählt.

Falls der Schnitt nicht leer ist, wird nach endlicher Zeit ein Wort aufgezählt, das im Schnitt enthalten ist. Damit ist also das Komplement von Π semi-entscheidbar.

(c) Wäre Π semi-entscheidbar, so wäre Π entscheidbar, da auch das Komplement semi-entscheidbar ist. Das ist aber ein Widerspruch zur Unentscheidbarkeit.

(d) Ähnlich wie in Teilaufgabe (b) wollen wir eine Turingmaschine $\mathcal{M}$ konstruieren, die zwei Turingmaschinen $\mathcal{M}_1$ und $\mathcal{M}_2$ genau dann akzeptiert, wenn $L(\mathcal{M}_1) \cap L(\mathcal{M}_2) \neq \emptyset$ gilt.

Da aber die gegebenen Turingmaschinen nicht notwendigerweise auf jeder Eingabe halten, können wir nicht einfach nacheinander alle Wörter in Σ^* aufzählen und $\mathcal{M}_1$ bzw. $\mathcal{M}_2$ darauf simulieren. Stattdessen simuliert $\mathcal{M}$ die beiden Turingmaschinen pseudoparallel für alle Wörter $w_1, w_2, \ldots \in \Sigma^*$ in kanonischer Reihenfolge. Das heißt, $\mathcal{M}$ simuliert zunächst einen Schritt von $\mathcal{M}_1$ auf w_1 und dann einen Schritt von $\mathcal{M}_2$ auf w_1. Danach simuliert $\mathcal{M}$ jeweils einen (weiteren) Schritt von $\mathcal{M}_1$ auf den Eingaben w_1 und w_2 und wiederholt das für $\mathcal{M}_2$, usw. Wenn $\mathcal{M}_1$ und $\mathcal{M}_2$ das gleiche Wort $w \in \Sigma^*$ akzeptieren, hält $\mathcal{M}$ und akzeptiert.

Falls der Schnitt nicht leer ist, werden durch die pseudoparallele Abarbeitung nach endlicher Zeit $\mathcal{M}_1$ und $\mathcal{M}_2$ auf einem Wort w simuliert, das im Schnitt enthalten ist.