FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
Professur für Didaktik der Informatik

Lösungs*vorschläge* zu den Staatsexamina:
Theoretische Informatik und Algorithmik

Herbst 2023

Inhaltsverzeichnis

1 Thema **2**
1.1 Teilaufgabe 1: Algorithmik . . . 2
1.2 Teilaufgabe 2: Theoretische Informatik . . . 2
1.2.1 Aufgabe 1: Reguläre Sprachen . . . 2
1.2.2 Aufgabe 2: Kontextfreie Sprachen . . . 4
1.2.3 Aufgabe 3: Chomsky-Hierarchie . . . 5
1.2.4 Aufgabe 4: Entscheidbarkeit . . . 7
1.2.5 Aufgabe 5: Komplexitätstheorie . . . 9

2 Thema **11**
2.1 Teilaufgabe 1: Algorithmik . . . 11
2.2 Teilaufgabe 2: Theoretische Informatik . . . 11
2.2.1 Aufgabe 1: Konstruktion . . . 11
2.2.2 Aufgabe 2: Minimale DEAs . . . 12
2.2.3 Aufgabe 3: Kontextfreie Grammatiken . . . 15
2.2.4 Aufgabe 4: Berechenbarkeit . . . 17

1 Thema

1.1 Teilaufgabe 1: Algorithmik

1.2 Teilaufgabe 2: Theoretische Informatik

1.2.1 Aufgabe 1: Reguläre Sprachen

(a) $L_a = \{w \in \Sigma^* | \nexists u, u' \in \Sigma^* : w = u(ab)u'\} = ((b|c)^*(a^*c)^*(b|c)^*)^*a^*$

Erklärung:
Alle Wörter in L_a setzen sich aus Teilworten zusammen, die eine zusammenhängendes Teilwort beliebiger Länge aus a's enthalten. Vor und nach dem a-Block dürfen beliebige Wörter aus b's und c's vorkommen (deshalb $(b|c)^*$ in regulären Ausdruck). Sobald ein a erscheint, dürfen weitere a's folgen (deshalb a^* in regulären Ausdruck) oder ein c (deshalb folgt auf a^* immer das c, außer am Ende des Wortes). Es müssen auch gar keine a's vorkommen (dehalb sind alle Bestandteile, in denen a's vorkommen, mit * versehen).

(b) $L_b = L(M) = a^*|(a^*ba^*ba^*ba^*ba^*)^*$.

Begründung:
Anfangs befinden wir uns in einem Endzustand, der beliebig viele weitere a's zulässt (deshalb a^* in regulären Ausdruck).
Verlässt man diesen Endzustand über ein b, muss ein Zyklus mit genau vier b's durchlaufen werden, zwischen denen jeweils beliebig viele a's liegen dürfen, bevor wieder der Endzustand erreicht wird (deshalb $a^*ba^*ba^*ba^*ba^*$ in regulären Ausdruck). Dieser Zyklus kann beliebig oft durchlaufen werden (deshalb das * bei der rechten Klammer im regulären Ausdruck).

(c) Zur besseren Übersicht wird zunächst die **Zustandsübergangs-Tabelle** des Automaten aufgestellt:

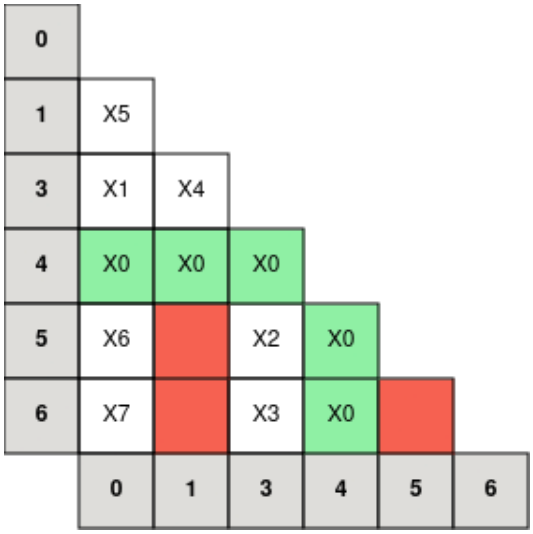
Zustand	**0**	**1**
0	1	3
1	0	6
2	1	4
3	4	4
4	4	5
5	0	5
6	0	5

Um die Erreichbarkeit aller Zustände zu überprüfen wird eine **Zeugentabelle** angefertigt:

Zustand	0	1	2	3	4	5	6
Zeuge	ϵ	0	-	1	11	111	01

Zustand 2 ist also nicht erreichbar, deshalb entfällt er im minimierten Automaten.

Nun wird das bekannte **Tabellenfüllverfahren** zur Bestimmung äquivalenter Zustände durchgeführt:



Da Zusatnd 4 der einzige Endzustand ist, werden zunächst alle Zustandspaare in der Tabelle, die 4 und einen weiteren Zustand enthalten, mit X_0 markiert.

Die weiteren Markierungen in der Tabelle gehen aus folgender Tabelle hervor, welche die Übergänge der Zustandspaare angibt:

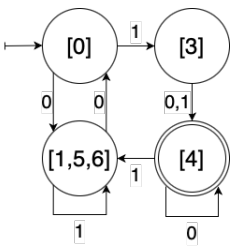
Zustandspaar	0	1	Erläuterung
(0,1)	(0,1)	(3,6)	Eingabe 1: (3,6) führt zu X3. Ergänze X5.
(0,3)	(1,4)	(3,4)	Eingabe 0: (1,4) führt zu X0. Ergänze X1.
(1,3)	(0,4)	(4,6)	Eingabe 0: (0,4) führt zu X0. Ergänze X4.
(0,5)	(0,1)	(3,5)	Eingabe 0: (3,5) führt zu X2. Ergänze X6.
(1,5)	(0,0)	(5,6)	
(3,5)	(0,4)	(4,5)	Eingabe 0: (0,4) führt zu X0. Ergänze X2.
(0,6)	(0,1)	(3,5)	Eingabe 1: (3,5) führt zu X2. Ergänze X7.
(1,6)	(0,0)	(5,6)	
(3,6)	(0,4)	(4,5)	Eingabe 0: (0,4) führt zu X0. Ergänze X3.
(6,5)	(0,0)	(5,5)	

Die Zuständspaare 1, 5 und 6 bleiben nach wiederholtem Durchlaufen und Überprüfen unmarkiert. Folglich liegen 1, 5 und 6 in der selben Äquivalentsklasse und bilden im minimietern Automaten den Zustand $[1,5,6]$. Insgesamt ergeben sich also folgende Äquivalenzklassen als Zustände für den minimierten Automaten: $[0], [1,5,6], [3], [4]$

Der minimierte Automat A' sieht also folgendermaßen aus:
$A' = (\{[0], [1,5,6], [3], [4]\}, \{0,1\}, \delta, [0], \{[4]\})$

mit δ:

Zustand	0	1
[0]	[1,5,6]	[3]
[1,5,6]	[0]	[1,5,6]
[3]	[4]	[4]
[4]	[4]	[1,5,6]

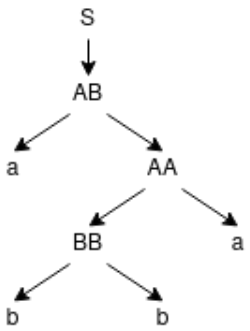


1.2.2 Aufgabe 2: Kontextfreie Sprachen

(a) Ja, das Wort „abba“ ist mit G ableitbar und liegt somit in $L(G)$, wie mit dem CYK-Algorithmus nachgewiesen werden kann:

a	b	b	a
A	B	B	A
S → AB	A → BB	B → BA	
B → AA	A → BB B → AA		
S → AB B → BA			

Der Ableitungsbaum lässt sich aus der Tabelle ablesen:



(b) i. Es gilt: $K \cdot R = \{k \circ r \mid \forall k \in K, r \in R\}$

Die regulären Sprachen sind eine Teilmenge der kontextfreien Sprachen. Also ist auch R kontextfrei. Kontextfreie Sprachen sind abgeschlossen unter *Verkettung*: Sei $G_1 = (N_1, \Sigma_1, P_1, S_1)$ die kontextfreie Grammatik, welche K erzeugt, und $G_2 = (N_2, \Sigma_2, P_2, S_2)$ die kontextfreie Grammatik, welche R erzeugt. Dann kann eine kontextfreie Grammatik G konstruiert werden, welche die Verkettung von K und R erzeugt:
Man ergänze dafür ein neues Startsymbol S, sowie eine neue Produktionsregel $S \to S_1 S_2$ und erhält $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$.
Da $L(G) = K \cdot R$ ist diese Sprache kontextfrei.

Die Aussage ist also wahr.

ii. L' ist bekanntermaßen nicht kontextfrei. Die Sprache $K = \{a^n \mid n \in \mathbb{N}_0\}$ über dem Alphabet $\Sigma = \{a, b, c\}$ ist regulär - denn es gilt $K = a^*$, also gibt es einen regulären Ausdruck für K - und somit auch kontextfrei.
Allerdings gilt auch für $K \circ L' = \{a^n b^m c^m \mid n, m \in \mathbb{N}_0\}$, das die Sprache regulär, mit Ausdruck: $K \circ L' = a^*(bc)^*$, und somit kontextfrei ist.

Dieses Gegenbeispiel beweist, dass die Aussage falsch ist.

1.2.3 Aufgabe 3: Chomsky-Hierarchie

(a) $L_1 = \{w \in L : vor\ jedem\ (\ stehen\ mehr\ [\ als\ ]\}$

Pumpinglemma für kontextfreie Sprachen
Ist L eine kontextfreie Sprache, so gilt:
$\exists p \in \mathbb{N} : \forall z \in L, |z| \geq p :$
$\exists u, v, w, x, y \in \Sigma^* : z = uvwxy$ mit

1. $|vx| \geq 1$
2. $|vwx| \leq p$
3. $\forall i \in \mathbb{N} : uv^iwx^iy \in L$

Nehmen wir an L_1 sei kontextfrei, dann können wir das Pumpinglemma anwenden und folgern:

Sei $p \in \mathbb{N}$ die Pumpingzahl. Wir wählen $z = [^p(]^p)] \in L_1$ mit $|z| = 2p + 2 > p$.
Aus $|vwx| \leq p$ und $|vx| \geq 1$ ergeben sich für die Form von vwx folgende Fälle:

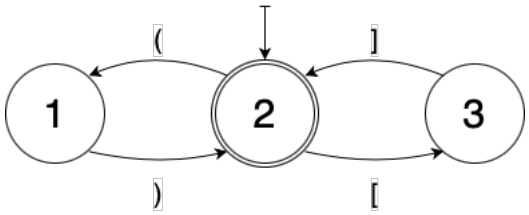
1. $v = [^k$ und $x = [^l$ mit $p \geq k + l \geq 1$
2. $v = [^k$ und $x = [^l(]^m$ mit $p - 1 \geq k + l + m \geq 0$
3. $v = [^k(]^l$ und $x =]^m$ mit $p - 1 \geq k + l + m \geq 0$
4. $v =]^l$ und $x =]^m$ mit $p \geq l + m \geq 1$
5. $v =]^l$ und $x =]^m)$ mit $p - 1 \geq l + m \geq 0$

In Fall 1 ist $uv^2wx^2y = [^{p+k+l}(]^p) \notin L_1$, da $p + k + l > p$ und somit liegt keine korrekte Klammerung mehr vor.
In Fall 2 ist $uv^2wx^2y = [^{p+k}(]^m[^l(]^p) \notin L_1$, da zwei (und nur eine) vorkommen. Somit liegt keine korrekte Klammerung vor.
In Fall 3 ist $uv^2wx^2y = [^p(]^l[^k(]^{p+m}) \notin L_1$, da zwei (und nur eine) vorkommen. Somit liegt keine korrekte Klammerung vor.
In Fall 4 ist $uv^2wx^2y = [^p(]^{p+l+m}) \notin L_1$, da $p + l + m > p$ und somit liegt keine korrekte Klammerung mehr vor.
In Fall 5 ist $uv^2wx^2y = [^p(]^{p+l})]^m) \notin L_1$, da zwei) und nur eine (vorkommen. Somit liegt keine korrekte Klammerung vor.

Das ist ein Widerspruch zur Annahme L_1 sei kontextfrei. $\Rightarrow L_1$ ist nicht kontextfrei $\Rightarrow L_1$ ist nicht regulär, da die regulären Sprachen eine Teilmenge der kontextfreien Sprachen bilden.

(b) $L_2 = \{w \in L : auf\ jede\ öffnende\ Klammer\ folgt\ direkt\ eine\ schließende\ Klammer\}$

Zu L_2 lässt sich folgender DEA bauen, der sie akzeptiert:



Somit ist L_2 regulär und damit auch kontextfrei, denn die regulären Sprachen sind eine Teilmenge der kontextfreien Sprachen.

1.2.4 Aufgabe 4: Entscheidbarkeit

(a) L_1 ist nicht entscheidbar.

TODO

Angenommen, L_1 wäre entscheidbar. Dann gäbe es eine Turingmaschine D, die für jede Eingabe $\langle M \rangle$ entscheiden kann, ob M auf $\langle N \rangle$ hält und dafür mindestens $2^{|\langle N \rangle|}$ Schritte benötigt. Dies würde bedeuten, dass D die Laufzeit von M auf $\langle N \rangle$ berechnen kann.

Allerdings ist bekannt, dass das Halteproblem unentscheidbar ist. Da das Halteproblem unentscheidbar ist, ist auch die Frage, ob M mindestens $2^{|\langle N \rangle|}$ Schritte benötigt, unentscheidbar, da dies eine strengere Bedingung ist, die das Halteproblem als Teilproblem enthält.

Da L_1 eine strengere Form des Halteproblems darstellt, muss auch L_1 unentscheidbar sein.

	Halteproblem	L_1
Eingabe	DTM M und Wort w	DTM M'
Ja-Instanzen	M hält auf w	DTM M' hält auf $\langle N \rangle$ nach $\geq 2^{\lvert\langle N \rangle\rvert}$ Schritten
Nein-Instanzen	M hält nicht auf w	DTM M' hält nicht auf $\langle N \rangle$ nach $\geq 2^{\lvert\langle N \rangle\rvert}$ Schritten

Beschreibung der Reduktion

Konstruiere M' wie folgt:

- $Eingabe = \epsilon$? Dann laufe 100 Schritte ohne das Band zu verändern, kehre zurück zum Ausgangspunkt, und verhalte dich wie M auf w.
- $Eingabe \neq \epsilon$? Verhalte dich wie M auf w.

(b) L_2 ist nicht entscheidbar.

Die Nicht-Entscheidbarkeit von L_2 wird durch eine Reduktion auf das bereits als unentscheidbar bekannte allgemeine Halteproblem $L_{halt} = \{\langle M \rangle w | M$ ist TM, die auf w hält$\}$ nachgewiesen. Dazu wird eine totale und berechenbare Funktion f benötigt, für die $f(w) \in L_2 \Leftrightarrow w \in L_{halt}$ gilt. Sei $f : \Sigma^* \to \Sigma^*$ definiert über:

TODO

$$f(w) = \begin{cases} \langle M' \rangle & \text{falls } w = \langle M \rangle v \text{ für TM } M \text{ und } v \in \Sigma^* \\ \varepsilon & \text{sonst} \end{cases}$$

Dabei ist M' eine TM, die auf $\langle N \rangle$ hält, genau dann wenn M auf v hält. Dabei lässt sich M' wie folgt konstruieren:

- Überprüfung der Eingabe w. Falls $w \neq \langle N \rangle$ in Endlosschleife laufen.
- Ansonsten löschen der Eingabe und schreiben von v aufs Band.
- Simulieren von M auf v. Falls M auf w hält, hält auch M'.

TODO $\Rightarrow$ siehe Bemerkung[1]

Die Funktion f ist offensichtlich total. Außerdem lässt sich f nach dem eben beschriebenen Vorgehen berechnen.

Es bleibt noch zu zeigen, dass $w \in L_{halt} \Leftrightarrow f(w) \in L_2$ gilt. Dies beweisen folgende Äquivalenzumformungen ($\forall w \in \Sigma^*$):

$$\begin{aligned} w \in L_{halt} &\Longleftrightarrow w = \langle M \rangle v \text{ mit TM } M \text{ hält auf } v \\ &\Leftrightarrow f(w) = \langle M' \rangle \text{ mit TM } M' \text{ hält auf } \langle N \rangle \\ &\Leftrightarrow f(w) \in L_2 \end{aligned}$$

Somit gilt $L_{halt} \leq L_2$ und da L_{halt} unentscheidbar ist, muss auch L_2 unentscheidbar sein.

(c) L_3 ist entscheidbar.

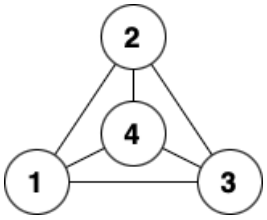
Da $L(N) = \Sigma^*$ muss N auf jedem $w \in \Sigma^*$ halten, um dieses zu akzeptieren. Folglich ist die Menge der Worte über Σ, auf denen N nicht hält, die leere Menge. Man kann L_3 also umschreiben zu:
$L_3 = \{\langle M \rangle | M \; hält \; auf \; mindestens \; einem \; w \in \varnothing\}$

Da es kein solches w gibt, kann es auch kein M geben, welches auf w hält. Folglich ist $L_3 = \varnothing$ und somit regulär und auch entscheidbar.

[1] Gilt das in beide Richtungen oder muss man die getrennt behandeln? Andere Richtung: M' hält auf $\langle N \rangle \rightarrow \langle M' \rangle \langle N \rangle$ als Eingabe.

1.2.5 Aufgabe 5: Komplexitätstheorie

(a) Ein Minimalbeispiel für einen Graphen, der eine Beinahe-3-Färbung aber keine 3-Färbung besitzt, sieht wie folgt aus:



Der Graph besitzt keine 3-Färbung, da Knoten $1, 2, 3$ und 4 alle jeweils paarweise durch Kanten verbunden sind. Wenn also Knoten $1, 2$ und 3 mit drei verschiedenen Farben belegt werden, muss für 4 ebenfalls eine dieser drei Farben gewählt werden und es entsteht eine Kante, die zwei Knoten mit gleicher Farbe verbindet. Folglich besitzt der Graph aber eine Beinahe-3-Färbung.

(b) Zunächst werden die genannten Entscheidungsprobleme in formale Sprachen übersetzt:

$L_{B3F} = \{c(G) \mid G = (V, E)$ ist Graph, für den eine Beinahe-3-Färbung existiert$\}$

$L_{3F} = \{c(G) \mid G = (V, E)$ ist Graph, für den eine 3-Färbung existiert$\}$

Um zu zeigen, dass das Problem $B3F$ in $\mathcal{NP}$ liegt, wird eine NTM beschrieben, welche L_{B3F} in polynomieller Zeit entscheidet:

1. Durchlaufe den Graphen Knoten für Knoten. ($O(n)$)
2. Wähle jeweils nichtdeterministisch die passende Farbe, sodass am Ende möglichst wenige Kanten aus zwei gleichfarbigen Knoten entstehen. ($O(1)$)
3. Gehe alle Kanten durch und überprüfe, ob höchstens eine Kante aus zwei gleichfarbigen Knoten gebildet wird. Wenn das der Fall ist akzeptiere, ansonsten halte und akzeptiere nicht. ($O(n)$)

Nun wird eine polynomielle Komplexitäts-Reduktion von $B3F$ auf $3F$ durchgeführt. Dazu wird eine totale und in polynomieller Zeit berechenbare Funktion f benötigt, die Probleme aus $3F$ auf Probleme aus $B3F$ abbildet:

Sei $f : \Sigma^* \to \Sigma^*$ definiert über

$$f(w) = \begin{cases} c(V \cup \{v_1, v_2, v_3, v_4\}, E \cup \{(v_1, v_2), (v_1, v_3), & , falls\ w = c(V, E)\ mit \\ (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}) & G = (V, E)\ ist\ Graph \\ 0 & , sonst \end{cases}$$

f ist offensichtlich total. Außerdem lässt sich eine DTM konstruieren, die f in polynomieller Laufzeit berechnet:

- Syntaxcheck, ob $w = c(V, E)$ mit $e_1, e_2 \in E$ und $G = (V, E)$ ist Graph ($O(n)$).
- Passendes anhängen der entsprechenden Knoten und Kanten an Eingabe zu $c(V \cup \{v_1, v_2, v_3, v_4\}, E \cup \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4)\})$ ($O(n^2)$)

Es bleibt noch zu zeigen, dass $w \in L_{3F} \Leftrightarrow f(w) \in L_{B3F}$ gilt. Dies beweisen folgende Äquivalenzumformungen ($\forall w \in \Sigma^*$):

$$
\begin{aligned}
w \in L_{3F} &\Longleftrightarrow w = c(V,E)\ mit\ Graph\ G=(V,E)\ besitzt\ eine\ \text{3-Färbung} \\
&\Leftrightarrow f(w) = c(V' := V \cup \{v_1,v_2,v_3,v_4\}, E' := E \cup \{(v_1,v_2),(v_1,v_3),(v_1,v_4),\\
&\quad\ (v_2,v_3),(v_2,v_4),(v_3,v_4)\})\ mit\ Graph\ G=(V,E)\ besitzt\ eine\ \text{3-Färbung} \\
&\Leftrightarrow f(w) = c(V',E')\ mit\ Graph\ G'=(V',E')\ besitzt\ eine\ \text{Beinahe-3-Färbung} \\
&\Leftrightarrow f(w) \in L_{B3F}
\end{aligned}
$$

Informelle Beschreibung: Minimalgraphen aus (a) zum Eingabegraphen hinzufügen. Somit wird eine 3-Färbung von G zu einer Beinahe-3-Färbung von G' und umgekehrt.

$B3F$ ist also $\mathcal{NP}$-hart und liegt in $\mathcal{NP}$. Somit ist $B3F$ $\mathcal{NP}$-vollständig.

2 Thema

2.1 Teilaufgabe 1: Algorithmik

2.2 Teilaufgabe 2: Theoretische Informatik

2.2.1 Aufgabe 1: Konstruktion

(a) Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein nichtdeterministischer endlicher Automat (NEA), sei $v \in \Sigma^*$ und sei $L = \{w \in \Sigma^* \mid w \in L(M) \text{ und } w \text{ enthält } v\}$. Wir konstruieren einen NEA $M' = (Q', \Sigma, \delta', q_0', F')$, für den $L(M') = L$ gilt, wie folgt:

- $Q' = Q \times \{0, 1, \ldots, k\}$, wobei k die Länge von v ist.
- δ' ist definiert durch:

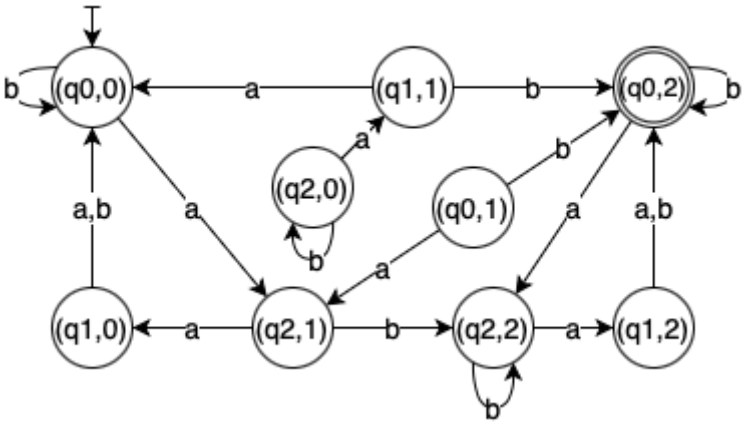
$$\delta'((q,i),\sigma) = \begin{cases} \{(q', i) \mid q' \in \delta(q, \sigma)\} & \text{falls } \sigma \neq v_{i+1} \text{ oder } i = k \\ \{(q', i+1) \mid q' \in \delta(q, \sigma)\} & \text{falls } \sigma = v_{i+1} \text{ und } i < k \end{cases}$$

- $q_0' = (q_0, 0)$
- $F' = \{(q, k) \mid q \in F\}$

Erklärung:
Die Zustände von M' sind Paare (q, i), wobei q ein Zustand von M ist und i die Position im Wort v darstellt. Wenn der Automat ein Zeichen σ liest, das nicht das nächste erwartete Zeichen von v ist, bleibt i unverändert. Wenn σ das nächste erwartete Zeichen von v ist, wird i inkrementiert. Sobald i die Länge von v erreicht hat (d.h. $i = k$), bleibt der Automat im Zustand k und simuliert weiterhin M.

Der Startzustand von M' ist $(q_0, 0)$, und die akzeptierenden Zustände sind die Paare (q, k), wobei q ein akzeptierender Zustand von M ist. Dies stellt sicher, dass das Wort v in w enthalten ist und dass w von M akzeptiert wird.

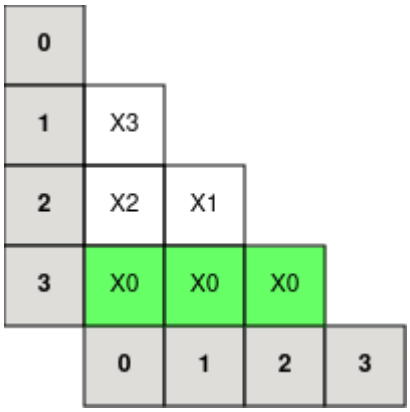
(b) Der Resultierende NFA sieht wie folgt aus (wobei $(q_0, 1), (q_1, 1)$ und $(q_2, 0)$ nicht erreichbar sind und nur der Vollständigkeit des Verfahrens halber mit aufgenommen wurden):



2.2.2 Aufgabe 2: Minimale DEAs

(a) Allgemein eignet sich für die Prüfung der Minimalität eines DEA das sogenannte „Tabellenfüllverfahren“(im Englischen *table filling algorithm* oder *table filling method*). Dabei wird eine Tabelle (mit allen erreichbaren Zuständen) aufgestellt, in der nicht äquivalente Zustandspaare gekennzeichnet werden. In dieser Tabelle wird zunächst in allen Zellen, deren zugehöriges Zustandspaar einen Endzustand und einen Nicht-Endzustand enthält, ein X_0 notiert - diese Paare können sicher nicht äquivalent sein.
Nun wird versucht, von den übrigen Zellen aus die bereits gefüllten Zellen durch Übergänge aus dem Automaten zu erreichen. Dies wird so lange wiederholt, bis sich keine Änderungen in der treppenförmigen Tabelle mehr ergeben. Wenn dann noch Zellen ungefüllt bleiben, bedeutet das, dass die Zustände des entsprechenden Zustandspaares äquivalent sind.
Alle zueinander äquivalenten Zustände liegen dann in einer Äquivalenzklasse, die nicht äquivalenten Zustände bilden jeweils eine eigene Äquivalenzklasse. Diese Äquivalenzklassen bilden die Zustände des minimierten DEA.

Für den ersten DEA bleibt keine Zelle der Tabelle ungefüllt, was bedeutet, dass hier keine äquivalenten Zustände vorliegen. Da jeder Zustand erreichbar ist, ist der Automat also bereits minimal:

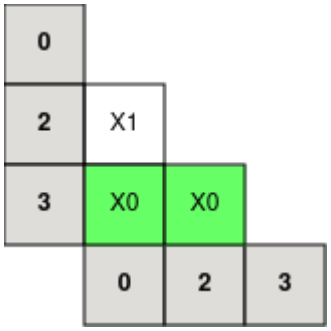


Zustandspaar	a	b	Erläuterung
(2,1)	(0,1)	(3,3)	Eingabe b: (3,3) führt zu X0. Ergänze X1.
(2,0)	(0,1)	(2,1)	Eingabe b: (2,1) führt zu X1. Ergänze X2.
(1,0)	(1,1)	(2,3)	Eingabe b: (2,3) führt zu X0. Ergänze X3.

(b) Eine Zeugentabelle zum Automaten zeigt, dass der Zustand 1 nicht erreichbar ist:

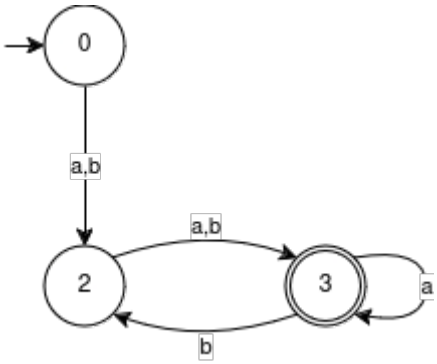
Zustand	**0**	**1**	**2**	**3**
Zeuge	ϵ	$-$	a	aa

Die übrigen Zustände sind wieder nicht äquivalent:

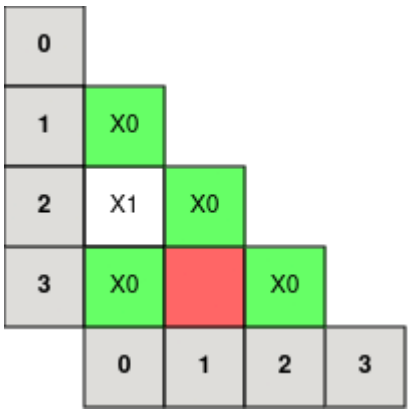


Zustandspaar	a	b	Erläuterung
(2,0)	(2,3)	(2,3)	Eingabe b: (2,3) führt zu X0. Ergänze X1.

Der minimierte DEA sieht dann folgendermaßen aus:

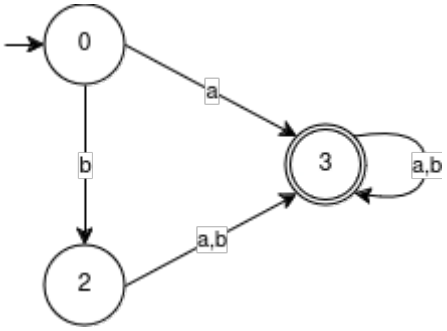


(c) Für diesen Automaten bleibt nach Durchführung des beschriebenen Tabellenfüllverfahrens eine Zelle ungefüllt. Dies ist der Nachweis, dass die Zustände 1 und 3 äquivalent zueinander sind und damit in der selben Äquivalenzklasse liegen.



Zustandspaar	a	b	Erläuterung
(2,0)	(1,3)	(2,3)	Eingabe b: (2,3) führt zu X0. Ergänze X1.
(3,1)	(3,3)	(1,1)	Kein Kreuz.

Der minimierte DEA besitzt nun noch einen Endzustand und kann folgendermaßen zusammengefasst werden:


0
a
b
3
a,b
a,b
2

2.2.3 Aufgabe 3: Kontextfreie Grammatiken

(a) Die Umwandlung von G in H erfolgt in vier Schritten:

1) Beseitigung von ϵ-Regeln

Alle Regeln der Form $A \to \epsilon$ werden beseitigt. Alle Produktionsregeln, die rechts ein Nichtterminal enthalten, das zuvor zu ϵ abgeleitet werden konnte, werden dafür ergänzt um eine Kopie *ohne* dieses Nichtterminalsymbol.
Da das leere Wort nicht in der von H erzeugten Sprache liegen soll, muss keine weitere Regel hinzugefügt werden.

Vorher			Nachher		
S	$\to$	aXY	S	$\to$	aXY | aY
X	$\to$	Y | ϵ	X	$\to$	Y
Y	$\to$	aS | b	Y	$\to$	aS | b

2) Beseitigung von Kettenregeln

Jede Produktionsregel der Form A $\to$ B mit A, B $\in$ V wird als „Kettenregel“bezeichnet. Dabei werden ggf. zunächst Zykel entfernt. Da hier keine Zykel vorliegen, muss nur noch die Regel X $\to$ Y angepasst werden. Dazu wird diese Regel entfernt und alle Regeln, deren linke Seite ein Y ist, um eine neue Regel mit X als linker Seite ergänzt.

Vorher			Nachher		
S	$\to$	aXY	S	$\to$	aXY
X	$\to$	Y	Y	$\to$	aS | b
Y	$\to$	aS | b	X	$\to$	aS | b

3) Ersetzen von Terminalzeichen

Jedes Terminalzeichen θ, das in Kombination mit anderen Symbolen auftritt, wird durch ein neues Nichtterminalzeichen Θ ersetzt. Zusätzlich wird die Produktionsregel $\Theta \to \theta$ ergänzt.

Vorher			Nachher		
S	$\to$	aXY	S	$\to$	AXY
Y	$\to$	aS | b	Y	$\to$	AS | b
X	$\to$	aS | b	X	$\to$	AS | b
			A	$\to$	a

4) Beseitigung von Nichtterminalketten

Alle Produktionsregeln, die in der rechten Seite mehr als zwei Nichtterminalzeichen aufweisen, werden durch einfügen zusätzlicher Nichtterminale in mehrere neue Regeln mit passender Form aufgeteilt.

Vorher	**Nachher**
S → AXY	S → AZ
Y → AS | b	Z → XY
X → AS | b	Y → AS | b
A → a	X → AS | b
	A → a

Die fertige Grammatik H in CNF, für die gilt $L(H) = L(G)\backslash\{\varepsilon\}$, sieht also folgendermaßen aus:

H = (V, Σ, P, S) mit V = {S, X, Y, Z, A}, Σ = {a, b} und P:

S → AZ
Z → XY
Y → AS | b
X → AS | b
A → a

(b) **Erklärung**
Zwei Einschränkungen gilt es zu beachten: CNF und $|V| \leq 5$. Außerdem darf die Grammatik einzig und allein das Wort „aaaaaaaaaaaa“ erzeugen.
Eine Nichtterminal wird benötigt, um mittels der Regel $E \to a$ letztendlich die Terminale zu erzeugen. Würden alle vier übrigen Variablen die Anzahl der erzeugten a's mit Regeln der Form $A \to BB$ verdoppeln, würde man $2^4 = 16$ a's erhalten. Um die Anzahl zu reduzieren wird im dritten Ableitungsschritt eine Regel der Form $C \to DE$ eingebaut. So werden im vorletzten Ableitungsschritt nur die D's erneut verdoppelt, während die E's direkt auf Terminalsymbole abgeleitet werden. Somit werden, wie erwünscht, nur $2^3 + 2^2 = 8 + 4 = 12$ a's erzeugt.

Die gesuchte Grammatik ist also $G = (V, \Sigma, P, A)$ mit $V = \{A, B, C, D, E\}$ und P:

A → BB
B → CC
C → DE
D → EE
E → a

(c) ————

TODO

————

Um die Anforderung der geraden Anzahl an Terminalzeichen zu erreichen, übernehmen wir aus P in P' nur diejenigen Produktionsregeln, die in Ableitungsbäume abgeleitet werden können, deren Blätteranzahl geradzahlig ist.

Die Einschränkung CNF bedeutet dabei, dass wir ein Nichtterminal entweder umwandeln müssen in ein Terminalzeichen, oder in zwei Nichtterminale. Es entfallen in P' also alle Produktionsregeln aus P, die eine beliebige Verzweigung zulassen würden, wie etwa solche der Form $X \to XY|YY$.

2.2.4 Aufgabe 4: Berechenbarkeit

(a) Ja, L ist semi-entscheidbar. M_1 und M_2 handelt es sich um Turing-Maschinen, daher erzeugen sie Typ-0-Sprachen.
Das Wortproblem für Typ-0-Sprachen ist semi-entscheidbar. Auch L ist eine Typ-0-Sprache, denn es ist sowohl eine Obermenge als auch eine Untermenge zweier anderer Typ-0-Sprachen. Damit ist auch L semi-entscheidbar.

(b) Ja L_1 L_2 ist auch semi-entscheidbar, denn wir können eine Turing-Maschine M' bauen, die $M(L_1)$ und $M(L_2)$ imitiert, und nur anhält, wenn sowohl $M(L_1)$ als auch $M(L_2)$ auf w anhalten.

(c) Ja, L' ist entscheidbar, es gibt also eine Turing-Maschine M', die für jedes Wort w entscheidet, ob es zu L' gehört oder nicht.

Dafür konstruieren wir $M'(L')$ wie folgt:

- Die Turing-Maschine erhält als Eingabe w = uv.
- M' prüft, ob sowohl u als auch v in L gehören, indem sie die Turing-Maschine für L auf beiden Teilen der Eingabe simuliert.
- Falls sowohl u als auch v in L sind, akzeptiert M' w, andernfalls verwirft sie w.

Da beide Teilwörter aus der entscheidbaren Sprache L stammen, wird auch M' für L' richtig entscheiden, ob w = uv in L' liegt oder nicht. Daher ist L' entscheidbar.

(d) Ja, auch L' ist unentscheidbar. L' besteht aus allen Präfixen von Wörtern in L. Wenn L unentscheidbar ist, gibt es keine Möglichkeit, zu entscheiden, ob ein gegebenes Wort u zu L' gehört oder nicht. Das würde nämlich bedeuten, dass wir für jedes Wort u ein entsprechendes Wort v finden müssten, so dass uv in L liegt, was wiederum bedeuten würde, dass L entscheidbar wäre.