Prof. Javier Esparza
Philipp Czerner, Martin Helfrich

Technische Universität München
Lehrstuhl für Theoretische Informatik

Einführung in die Theoretische Informatik

Sommersemester 2021 – Hausaufgabenblatt 4

- Beachten Sie die Abgabemodalitäten auf der Vorlesungswebsite!
- Sei $\Phi := \{\{1\}, \{2\}, \{3,4\}, \{5,6\}, \{7\}\}$. Nach dem Abgabedatum werden wir für jede Menge $A \in \Phi$ eine zufällige Aufgabe $a \in A$ wählen und korrigieren.
- Es werden diese Aufgaben korrigiert: H4.1, H4.2, H4.3, H4.6, H4.7
- Wenn Sie einen Beweis aufstellen, von dem Sie wissen, das einzelne Schritte problematisch oder unvollständig sind, merken Sie dies bitte in Ihrer Lösung an, damit wir das bei der Korrektur positiv berücksichtigen können.

Aufgabe H4.1. (*vero nihil verius*) 0.5+0.5+0.5+0.5+1 Punkte

Bestimmen Sie für folgende Aussagen, ob sie wahr oder falsch sind. Geben sie eine kurze Begründung für wahre Aussagen an und ein Gegenbeispiel für falsche.

Sei $\Sigma \neq \emptyset$ ein Alphabet, $L_1, L_2 \subseteq \Sigma^*$ reguläre Sprachen, und $L_3, L_4 \subseteq \Sigma^*$ nicht-reguläre Sprachen.

(a) $(L_1 \cap L_2)^2$ ist regulär.

(b) $L_3 \cup L_4$ ist nicht regulär.

(c) L_4 erfüllt die Eigenschaft des Pumping-Lemmas nicht.

(d) Jede Teilmenge $L \subseteq L_1$ ist regulär.

(e) $(L_3)^*$ ist regulär.

Lösungsskizze. Anmerkung: Diese Aufgabe zielt darauf ab, die Abschlusseigenschaften regulärer Sprachen zu wiederholen. Durch Zitat des entsprechenden Satzes sind sie daher in einer Zeile gelöst. In dieser Lösung geben wir auch die Begründungen/Konstruktionen an, woher die Abschlusseigenschaften kommen.

(a) Wahr. Da L_1 und L_2 regulär sind, gibt es DFAs N_1 und N_2 mit $L(N_1) = L_1$ und $L(N_2) = L_2$. Die Produktkonstruktion liefert einen Automaten $N_1 \times N_2$ für $L_1 \cap L_2$, somit ist $L_1 \cap L_2$ regulär. Damit gibt es einen regulären Ausdruck r mit $L(r) = L_1 \cap L_2$. Dann gilt $L(rr) = (L_1 \cap L_2)^2$, somit ist die geforderte Sprache regulär.

(b) Falsch. Sei L_3 eine beliebige nicht-reguläre Sprache. Wähle $L_4 = \bar{L}_3$. Wir behaupten L_4 ist nicht-regulär. Angenommen L_4 wäre regulär. Dann gibt es einen DFA für L_4 und das Komplement dieses DFAs berechnet daher die Sprache L_3, die nicht-regulär ist, Widerspruch. Also ist L_4 nicht-regulär. Aber $L_3 \cup L_4 = \Sigma^*$ ist regulär.

(c) Falsch. In der Vorlesung wurde erwähnt, dass auch nicht-reguläre Sprachen das Pumping-Lemma erfüllen können. In Aufgabe H3.2 wurde außerdem ein konkretes Beispiel angegeben.

(d) Falsch. Gegenbeispiel: Wähle $\Sigma = \{a, b\}$, $L_1 := \Sigma^*$ und wähle für L eine beliebige nicht-reguläre Sprache, zum Beispiel $\{a^n b^n : n \in \mathbb{N}\}$. Dann ist L_1 regulär und $L \subseteq L_1$, aber L ist nicht-regulär.

(e) Falsch. Gegenbeispiel: Wähle $\Sigma = \{a, b, c\}$, $L_3 := \{a^n b^n c : n \in \mathbb{N}\}$. Wenn $(L_3)^*$ regulär wäre, dann wäre nach Abschlusseigenschaften auch $(L_3)^* \cap L(a^*b^*c) = L_3$ regulär, ein Widerspruch.

Anmerkung: Bei der letzten Teilaufgabe gibt es gute und schlechte Wahlen von L_3. Die Sprache $L = \{w = w^R : w \in \Sigma^*\}$ der Palindrome zu wählen, würde zu $L^* = \Sigma^*$ führen, da jedes Zeichen einzeln ein Palindrom ist und daher jedes Wort trivial eine Konkatenation von Palindromen. Das Problem ist, dass Palindrome mit allen möglichen Buchstaben anfangen und aufhören können. Wenn wir also nur ein Wort wie *aabbbaaabb* haben, gibt es viele Möglichkeiten dies in Teil-Palindrome zu zerlegen. Wir könnten z.B. jedes Zeichen für sich lesen, oder $a(abbba)(aa)bb$ als Teil-Palindrome nehmen. Unsere Wahl von L_3 umgeht dies: Das Wortende ist in L_3 mit einem c markiert, sodass ein Wort aus L_3^* sich leicht in seine Teilwörter zerlegen lässt.

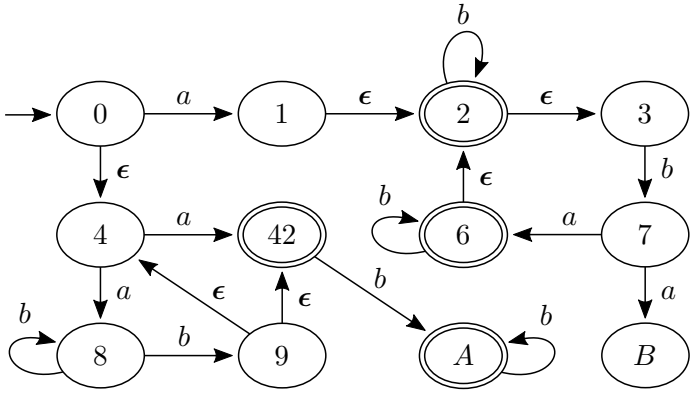
Aufgabe H4.2. (*in nuce*) 1+1 Punkte

(a) Vereinfachen Sie den folgenden ϵ-NFA M, konstruieren Sie also einen äquivalenten ϵ-NFA mit höchstens 6 Zuständen.

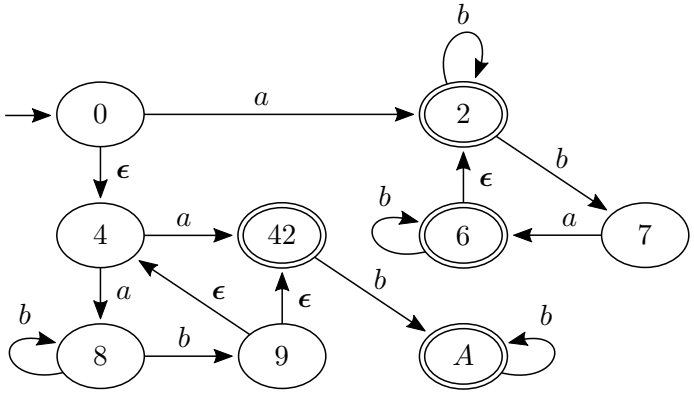
(b) Finden Sie einen regulären Ausdruck r mit $|r| \leq 6$ und $L(r) = L(M)$.

Beschreiben Sie bei beiden Teilaufgaben ihr Vorgehen.

Hinweise: Bei der Länge eines regulären Ausdrucks zählen wir Klammern nicht, die Länge von $(ab \mid \epsilon)^*$ wäre also 5. Zur Lösung dieser Aufgabe sind keine Kenntnisse über minimale DFAs vonnöten.

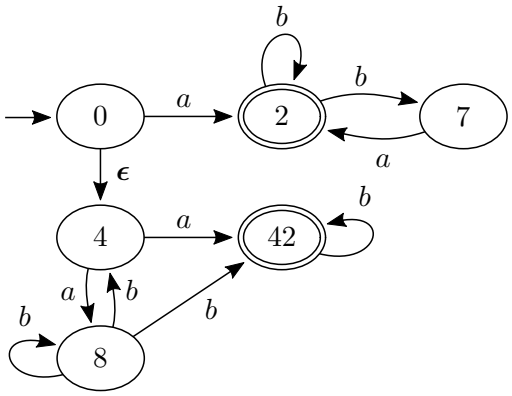


Lösungsskizze. **(a)** Zuerst entfernen wir die unnötigen Zustände 1, 3 und B.

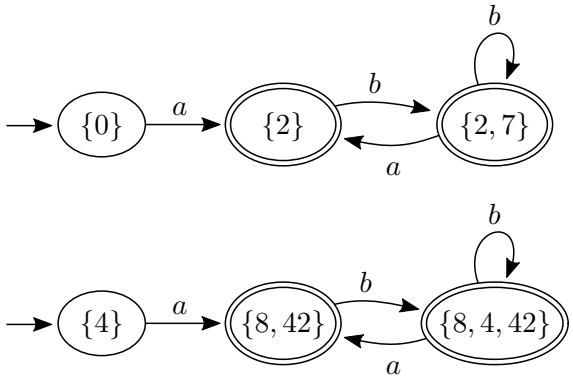


Da Zustand 6 lediglich beliebig viele b einliest und dann zu Zustand 2 wechselt, der ebenfalls beliebig viele b einliest (und beide Zustände akzeptierend sind), lässt sich die

Kante von 6 zu 2 kollabieren. Von 42 aus wird jede Folge an b akzeptiert, auch das leere Wort, also können wir die Kante von 42 nach A kollabieren. Schließlich können wir Zustand 9 entfernen, indem wir Kanten von 8 direkt zu 4 und 12 erstellen.



(b) Die oberen Zustände (also $\{0, 2, 7\}$) des Automaten akzeptieren genau $a(b \mid ba)^*$. Dies sind die Wörter, die mit einem a beginnen und aa nicht enthalten. An dieser Stelle kann man auch durch clevere Umformungen darauf kommen, dass der untere Teil des Automaten genau die gleiche Sprache akzeptiert – falls man dies allerdings nicht bemerkt, bietet sich eine Potenzmengenkonstruktion an, um die beiden Teile zu vereinfachen.

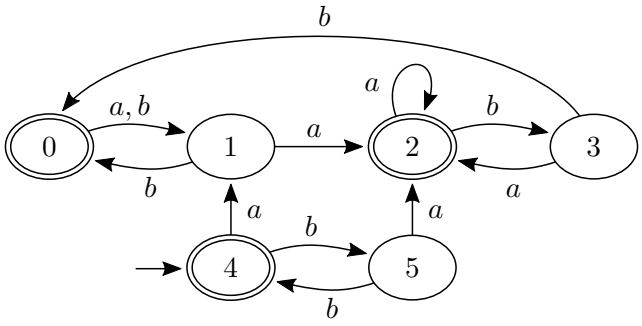


Nicht gezeichnete Kanten führen zu einem impliziten Fangzustand. Da beide DFAs identisch sind, müssen auch beide Teile des ϵ-NFAs die gleiche Sprache akzeptieren. Somit können wir Zustände $\{4, 8, 42\}$ löschen, und erhalten $a(b \mid ba)^*$ als regulären Ausdruck äquivalent zu M mit Länge 6.

Aufgabe H4.3. (*a maiore ad minus*) 2+1 Punkte

(a) Bestimmen Sie für folgenden DFA M die Äquivalenzklassen von $\equiv_M$ mit der Erweiterung des in der Vorlesung vorgestellten Verfahrens aus Ü4.4. Geben Sie also insbesondere für jedes Paar an Zuständen (q, r), das sie „markieren“, ein Wort w an, das sie unterscheidet (d.h. von $\delta(q, w), \delta(r, w)$ ist genau einer akzeptierend). Es steht ihnen frei, auch die optimierte Version des Verfahrens mit Abhängigkeitsanalyse zu verwenden.

(b) Erzeugen Sie einen minimalen DFA äquivalent zu M, indem sie die äquivalenten Zustände von M kollabieren.

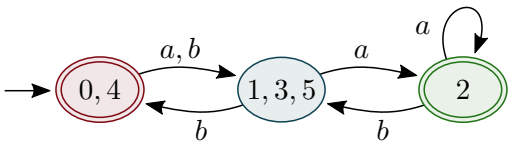


Lösungsskizze.

0					0				
$\boldsymbol{\varepsilon}$	1				ε	1			
	$\boldsymbol{\varepsilon}$	2			$\boldsymbol{a}$	ε	2		
$\boldsymbol{\varepsilon}$		$\boldsymbol{\varepsilon}$	3		ε		ε	3	
	$\boldsymbol{\varepsilon}$		$\boldsymbol{\varepsilon}$	4		ε	$\boldsymbol{a}$	ε	4
$\boldsymbol{\varepsilon}$		$\boldsymbol{\varepsilon}$		$\boldsymbol{\varepsilon}$ 5	ε		ε		ε 5
0									
ε	1								
a	ε	2							
ε		ε	3						
	ε	a	ε	4					
ε		ε		ε 5					

(a) Zuerst markieren wir alle Paare die genau einen akzeptierenden Zustand besitzen als unterschiedlich. Danach gehen wir alle verbleibenden Paare durch: $(0,2) \xrightarrow{a} (1,2)$ und $(1,2)$ ist markiert mit ε, also wird auch $(0,2)$ markiert, und zwar mit a. $(1,3) \xrightarrow{a} (2,2)$ und $(1,3) \xrightarrow{b} (0,0)$, also bleibt $(1,3)$ unmarkiert (und wird es auch immer sein). $(2,4) \xrightarrow{a} (2,1)$, also wird $(2,4)$ markiert, ebenfalls mit a. $(3,5) \xrightarrow{a} (2,2)$ und $(3,5) \xrightarrow{b} (0,4)$, also bleibt $(3,5)$ unmarkiert. $(0,4) \xrightarrow{a} (1,1)$ und $(0,4) \xrightarrow{b} (1,5)$, also bleibt $(0,4)$ unmarkiert. Schließlich gilt $(1,5) \xrightarrow{a} (2,2)$ und $(1,5) \xrightarrow{b} (0,4)$, und $(1,5)$ bleibt auch unmarkiert. In der nächsten Iteration gibt es keine Veränderungen. Die Äquivalenzklassen sind also $\{\{0,4\},\{1,3,5\},\{2\}\}$.

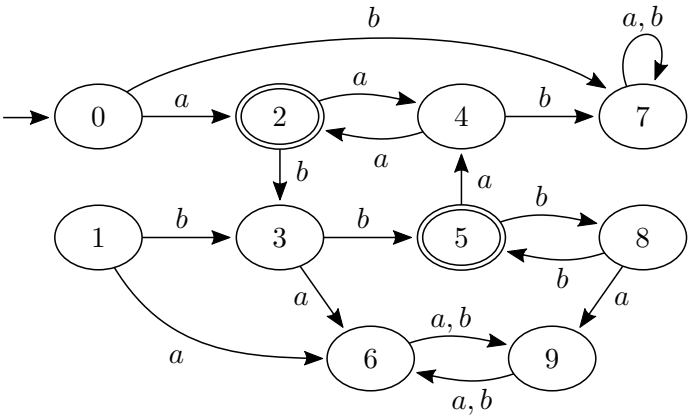
(b) Der minimierte DFA sieht folgendermaßen aus:



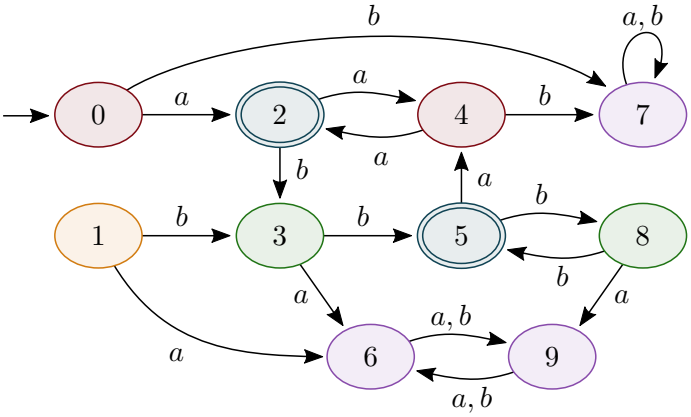
Aufgabe H4.4. (*a maximo ad minimum*) 2+1 Punkte

(a) Bestimmen Sie für folgenden DFA M' die Äquivalenzklassen von $\equiv_{M'}$. Sie müssen hierzu nicht nach dem in der Vorlesung vorgestellten Verfahren vorgehen.

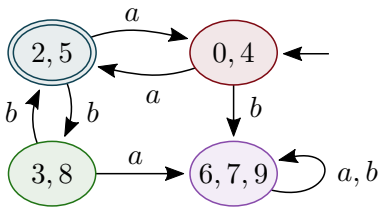
(b) Minimieren Sie nun M' mit dem Minimierungsalgorithmus aus der Vorlesung und geben Sie einen regulären Ausdruck für $L(M')$ an.



Lösungsskizze. **(a)** Die Äquivalenzklassen sind $\{\{0,4\},\{1\},\{2,5\},\{3,8\},\{6,7,9\}\}$.



(b) Zuerst eliminieren wir den nicht-erreichbaren Zustand 1. Danach kollabieren wir äquivalente Zustände und erhalten folgenden Automaten.



Zustand 7 ist ein Fangzustand, der DFA akzeptiert also die Sprache, die von dem regulären Ausdruck $a(aa \mid bb)^*$ erzeugt wird.

Aufgabe H4.5. (*ejusdem generis*) 1+3 Punkte

(a) Sei $\Sigma := \{a\}$, $k \in \mathbb{N}_{>0}$ beliebig und $L := \{a^{ik} : i \in \mathbb{N}\}$ die Sprache der Wörter deren Länge ein Vielfaches von k ist. Bestimmen sie die Äquivalenzklassen von $\equiv_L$ (s. Definition 3.55), und zeigen Sie so, dass der minimale DFA für L genau k Zustände hat.

(b) Sei $r := ab^* \mid ba^*$ ein regulärer Ausdruck über $\Sigma := \{a, b\}$. Konstruieren Sie den kanonischen Minimalautomaten für $L(r)$ direkt nach Definition 3.59. Beschriften Sie hierfür jeden Zustand mit einem regulären Ausdruck, dessen erzeugte Sprache der Äquivalenzklasse des Zustandes entspricht. (Der Startzustand wäre z.B. mit

ϵ beschriftet.) Beschreiben Sie ihr Vorgehen. Sie müssen nicht beweisen, dass Sie jeder Äquivalenzklasse einen passenden regulären Ausdruck zugeordnet haben.

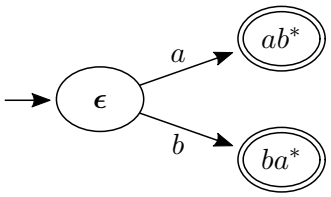
Achtung: Wir betrachten die Äquivalenzklassen bezüglich $\equiv_{L(r)}$, nicht die der Zustände des Automaten!

Lösungsskizze. **(a)** Zwei Wörter u, v sind äquivalent genau dann, wenn $|u| \equiv |v| \pmod{k}$. Die Äquivalenzklassen von $\equiv_L$ sind also

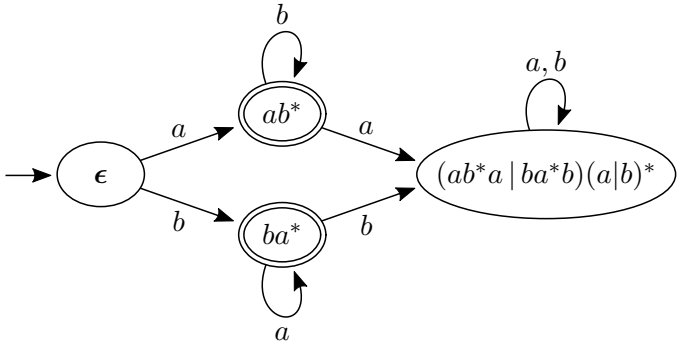
$$[a^j]_{\equiv_L} = \{a^{j+ik} : i \in \mathbb{N}\} \quad \text{für } j \in \{0, ..., k-1\}$$

(b) (Wie üblich ergibt sich die Äquivalenzrelation implizit aus dem Kontext und wir schreiben $[\cdot]$ statt $[\cdot]_{\equiv_{L(r)}}$.) Zuerst müssen wir uns davon überzeugen, dass $[\varepsilon] = \{\varepsilon\}$ tatsächlich gilt, wie von der Aufgabenstellung behauptet. Zur Erinnerung: $[\varepsilon]$ enthält die Wörter w, sodass für jedes Wort u das Wort wu genau dann in L ist, wenn u es auch ist. Da $ab \in L$, muss somit auch $wab \in L$ gelten – aber jedes Wort in L hat genau ein a oder genau ein b, also gilt $w = \varepsilon$.

Vom Startzustand ϵ ausgehend müssen wir nun die Äquivalenzklassen $[a]$ und $[b]$ bestimmen. An das Wort a lassen sich genau die Wörter in b^* anhängen (um ein Wort in L zu erhalten), die Äquivalenzklasse ist also ab^*. Analog dazu ist gilt $[b] = L(ba^*)$. Außerdem gilt $\varepsilon \in [a], [b]$, beide Zustände sind somit final. Der Automat sieht nun wie folgt aus:



Wir fahren mit Zustand $[a] = L(ab^*)$ fort. Da $ab \in L(ab^*)$, gilt $[ab] = [a]$ und es gibt eine b-Schleife von diesem Zustand aus. Für aa gilt, dass es kein Wort w gibt, sodass $aaw \in L$, die Äquivalenzklasse $[aa]$ enthält also genau die Wörter, die *nicht* Präfix eines Wortes in L sind. Das sind genau die Wörter, bei denen der erste Buchstabe ein zweites Mal im Wort vorkommt, also $ab^*a(a \mid b)^*$ und $ba^*b(a \mid b)^*$. Beachten Sie, dass beide Ausdrücke mit $(a \mid b)^*$ enden, da ein schlechtes Präfix nicht gut werden kann, indem man Zeichen anhängt. Für $[ba]$ und $[bb]$ ist die Situation dann analog.



Aufgabe H4.6. (*sic parvis magna*) 1+2+1 Punkte

Dora hat ihren Kindergartenprofessor noch einmal gefragt, und weiß nun, wie die Produkt-Konstruktion funktioniert. Jetzt ist ihr aber aufgefallen, dass der entstandene Produktautomat nicht minimal sein muss! Entrüstet möchte sie zu ihrem Kindergartenprofessor stampfen, ihn mit Sand bewerfen, und sich darüber beschweren, dass er ihr ein offensichtlich schlechtes Verfahren beigebracht hat. Können Sie Dora beschwichtigen, indem Sie demonstrieren, dass die ihr bekannten Verfahren, um Mengenoperationen auf DFAs auszuführen, zumindest manchmal ein bestmögliches Ergebnis liefern?

(a) Sei L eine reguläre Sprache und $M = (Q, \Sigma, \delta, q_0, F)$ ein minimaler DFA für L. Zeigen Sie, dass es einen minimalen DFA für $\overline{L}$ mit $|Q|$ Zuständen gibt.

(b) Zeigen Sie, dass es für jede Zahl $n \in N$ zwei DFAs $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ gibt, sodass $|Q_1|, |Q_2| \geq n$ und der minimale Automat für $L(M_1) \cap L(M_2)$ genau $|Q_1| \cdot |Q_2|$ Zustände hat.

(c) Beweisen Sie, dass es für jede Zahl $n \in N$ zwei DFAs $M_3 = (Q_3, \Sigma, \delta_3, q_{03}, F_3)$ und $M_4 = (Q_4, \Sigma, \delta_4, q_{04}, F_4)$ gibt, sodass $|Q_3|, |Q_4| \geq n$ und der minimale Automat für $L(M_3) \cup L(M_4)$ genau $|Q_3| \cdot |Q_4|$ Zustände hat.

Hinweis: Es mag hilfreich sein, Ergebnisse aus Aufgabe H4.5 zu verwenden.

Lösungsskizze.

(a) Das Verfahren zur Komplementbildung von DFAs, das wir kennen, macht jeden Finalzustand normal und umgekehrt. (Die neuen Finalzustände sind also $Q \setminus F$.) Dies verändert die Anzahl der Zustände nicht und wir wissen schon mal, dass es einen DFA für $\overline{L}$ mit $|Q|$ Zuständen gibt. Wenn es einen DFA $M' = (Q', \Sigma, \delta', q_0', F')$ für $\overline{L}$ mit $|Q'| < |Q|$, also echt weniger als $|Q|$ Zuständen, gäbe, könnten wir dessen Komplement bilden und würden einen DFA für L mit $|Q'|$ Zuständen erhalten. Aber M war bereits minimal, dies ist also nicht möglich. Somit kann kein Automat für $\overline{L}$ weniger als $|Q|$ Zustände haben, und das Komplement von M ist bereits minimal.

(b) Wir verwenden, dem Hinweis folgend, Aufgabe H4.5 (a) und betrachten die Sprachen $L_1 := \{a^{in} : i \in \mathbb{N}\}$ und $L_2 := \{a^{i(n+1)} : i \in \mathbb{N}\}$. Da n und $n+1$ aufeinanderfolgende natürliche Zahlen sind, sind sie teilerfremd, und damit ist ihr kleinstes gemeinsames Vielfaches $n(n+1)$. Hieraus folgt $L_1 \cap L_2 = \{a^{in(n+1)} : i \in \mathbb{N}\}$. Nach H4.5 (a) besitzen die minimalen DFAs für L_1 bzw. L_2 damit n bzw. $n+1$ Zustände, und der minimale DFA für $L_1 \cap L_2$ besitzt $n(n+1)$ Zustände, wie zu zeigen war. (Es ist also M_1 der minimale DFA für L_1, und M_2 für L_2.)

Anmerkung: Statt n und $n+1$ hätten wir auch genauso gut zwei beliebige Primzahlen $p_1, p_2 \geq n$ wählen können.

(c) Wir wählen L_1, L_2 genau so wie in in Teilaufgabe (b), und definieren M_{i+2} als minimalen DFA für das *Komplement* von L_i für $i = 1, 2$, d.h. $L(M_3) = \overline{L_1}$ und $L(M_4) = \overline{L_2}$. Nach Teilaufgabe (a) gilt $|Q_1| = |Q_3|$ und $|Q_2| = |Q_4|$, da der minimale DFA einer Sprache und ihres Komplements gleich groß sind. Außerdem gilt nach De Morgan

$$L(M_3) \cup L(M_4) = \overline{\overline{L(M_3)} \cap \overline{L(M_4)}} = \overline{L_1 \cap L_2}$$

Nach (b) hat der minimale DFA für $L_1 \cap L_2$ genau $|Q_1| \cdot |Q_2| = |Q_3| \cdot |Q_4|$ Zustände, und somit nach (a) auch der minimale DFA für $\overline{L_1 \cap L_2} = L(M_3) \cup L(M_4)$.

Bonusaufgabe H4.7. (*inveniet quod quisque velit*) 3 Bonuspunkte

Der Superschurke Dr. Evilsparza hat jüngst ein Pharmaunternehmen (Evillest Evilness *Inc.*) gegründet, und behauptet nun, einen Impfstoff gegen schlechtes Wetter entwickelt zu haben. Die Öffentlichkeit ist begeistert – Sie trauen dem angeblich reformierten Sünder allerdings nicht und versuchen, in der DNA-Sequenz nach Hinweisen zu suchen.

Bekanntermaßen werden in der DNA vier verschiedene Basen kodiert: Evalcyclohexandifluorit, Isophoronditrigocyanat, Vinylcyclohexenquadroxid, und Levomethagameorphan. Diese werden mit ihrem Initialbuchstaben abgekürzt, sodass letztendlich ein Wort über dem Alphabet $\Sigma := \{e, i, l, v\}$ entsteht. Sie sind in Besitz einer Sequenz, mit der Dr. Evilsparza in 2006 versuchte, die gesamte Studierendenschaft zu mathematischen Zombies zu wandeln, um die Universitätsherrschaft an sich zu reißen. Sie vermuten, dass er diese Sequenz wieder verwendet – aber Sie müssen Beweise finden, und die Zeit drängt!

Unter diesem Link finden Sie sowohl die Impfstoff-Sequenz w, als auch die Zombie-Sequenz s. Bestimmen Sie, ob s in w enthalten ist, und geben sie den kleinsten Index i an, sodass $w_1w_2...w_i$ bereits s enthält. Beschreiben Sie ihr Vorgehen.

Hinweise: Verwenden Sie zur Lösung dieser Aufgabe einen Computer. Beschreiben Sie bitte Ihren Ansatz in *natürlicher Sprache* und illustrieren die wesentlichen Schritte ihrer Lösung mit geeigneten Codefragmenten. Sie können (müssen aber nicht), ihrer Lösung ihren vollständigen Programmcode beifügen, jedoch steht es den Korrektoren frei, diesen zu ignorieren. Wenn Sie existierende Algorithmen verwenden, die nicht Teil der Vorlesung sind, beschreiben Sie bitte ausführlich, weshalb und auf welche Weise diese funktionieren.

Wir sind, wie immer, an Lösungen interessiert, die sich verallgemeinern lassen. Ihre Lösung sollte also für beliebige Wörter w und s funktionieren. Zusätzlich soll ihre Lösung effizient sein (im Bezug auf die theoretische Komplexität des Algorithmus), und innerhalb von wenigen Sekunden (aber auf jeden Fall innerhalb von einer Minute) terminieren. Sie dürfen Standard-Datenstrukturen (wie etwa Hashtabellen oder binäre Bäume) verwenden, ansonsten beschränken Sie sich bitte auf grundlegende Funktionalitäten verbreiteter Programmiersprachen.

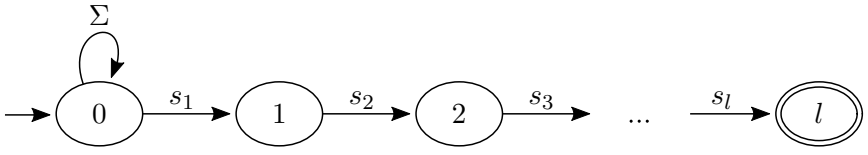
Highscores: Die Musterlösung wurde nicht optimiert und terminiert in etwa 750ms. Falls ihr Programm schneller ist, können Sie gerne eine Mail an Philipp Czerner mit dem Betreff „THEO H4.7“ schreiben, in der Sie ihr Programm und Anweisungen, es auszuführen, anhängen. Wir veröffentlichen dann eine Bestenliste auf Zulip. (Um die Zeiten für die Bestenliste zu ermitteln, werden wir etwas andere $w, s \in \Sigma^*$ verwenden.)

Lösungsskizze. Zuerst lesen wir die beiden Dateien ein:

```
w = open('impfstoff.txt', 'r').read()
s = open('zombie.txt', 'r').read() + '.'
```

An s hängen wir dabei noch ein `$` an, da dies den folgenden Code vereinfachen wird.

Wir erzeugen zunächst wir einen NFA M, um s (mit Länge $l := |s|$) zu erkennen:



In Mengenschreibweise würden wir $M = (Q, \Sigma, \delta, q_0, F)$ also definieren, indem wir Zustände $Q := \{0, ..., l\}$ wählen, $q_0 := 0$, $F := \{l\}$, und Übergänge

$((0, c), 0)$	für $c \in \Sigma$
$((i-1, s_i), i)$	für $i \in \{1, ..., l\}$

Diesen Automaten konvertieren wir nun zu einem DFA $M' = (Q', \Sigma, \delta', q_0', F')$, indem wir eine Potenzmengenkonstruktion anwenden. Prinzipiell könnte M' bis zu 2^{1000} Zustände haben, tatsächlich hat M' aber nur $|s|+1$ Zustände. (Dies lässt sich mit der Beobachtung zeigen, dass $q := \delta(\{q_0\}, s_1 s_2 ... s_i)$ der einzige Zustand von M' ist, der $i \in q, i+1, i+2, ... \notin q$ erfüllt. Für weitere Details seien interessierte Studierende Vorlesung Automatentheorie verwiesen.) Die Potenzmengenkonstruktion lässt sich folgendermaßen implementieren.

```
states = {q_0}
transitions = {}
q_0 = (0,)
final = set()
workset = [q_0]
while workset:
    q = workset.pop()
    for c in 'eilv':
        q_next = tuple([0] + [i+1 for i in q if s[i] == c])
        if q_next not in states:
            states.add(q_next)
            if len(s)-1 in q_next: final.add(q_next)
            workset.append(q_next)
        transitions[q,c] = q_next
```

Dies ist ein Workset-Algorithmus: Die Zustände, die wir noch nicht abgearbeitet haben, befinden sich in `workset`. Wir nehmen uns wiederholt einen Zustand aus dem Workset heraus, in erzeugen dessen ausgehende Transitionen. Immer wenn wir einen neuen Zustand erreichen, fügen wir ihn in das Workset ein.

Schlussendlich erhalten wir

- `states`: Q', gespeichert in einem `set`, damit wir effizient überprüfen können, ob ein Element enthalten ist.
- `transitions`: δ', als `dict`, bildet $(q, c) \in Q' \times \Sigma'$ Paare auf $\delta'(q, c)$ ab.
- `q_0`: Der Startzustand q_0'.
- `final`: Die Finalzustände F', wieder als `set`.

Hier werden die Zustände direkt als Tupel dargestellt, z.B. wäre $\{0, 3, 8\} \in Q'$ das Python-Tupel `(0,3,8)`. Es gibt zwar nur $|s| + 1$ Zustände in M', aber diese können auch bis zu $|s| + 1$ Elemente enthalten. Das macht es ineffizient, direkt mit den Tupeln zu arbeiten. Stattdessen benennen wir die Zustände jetzt um, sodass sie wieder Zahlen $0, ..., |s|$ sind.

```
M = {q: i for i, q in enumerate(states)}
transitions = {(M[q],c): M[r] for (q,c),r in transitions.items()}
q_0 = M[q_0]
final = {M[i] for i in final}
```

Hierzu erstellen wir ein `dict`, also `M`, das Zustände in `states` auf Zahlen in $0, ..., |s|$ abbildet. Das verwenden wir dann, um die Zustände in `transitions`, `q_0` und `final` umzubenennen. Schließlich müssen wir nur den DFA auf dem Wort w ausführen.

```
q = q_0
for i, c in enumerate(w):
    q = transitions[q,c]
    if q in final: print(i+1); break
```

Das Programm gibt 4026864 aus, also ist s bereits in $w_1 w_2 ... w_{4026864}$ enthalten.