Formale Systeme, Automaten, Prozesse, SS 2020
Prof. Dr. P. Rossmanith
J. Dreier, T. Hartmann, H. Lotze, D. Mock

Datum: 4.5.2020

Übungsblatt mit Lösungen 04

Aufgabe T12

Richtig oder falsch?

a) Es gibt eine reguläre Sprache, die von einem NFA mit 10 Zuständen akzeptiert wird, aber von keinem DFA mit 100 Zuständen.

b) Es gibt eine reguläre Sprache, die von einem NFA mit 10 Zuständen akzeptiert wird, aber von keinem DFA mit 1217 Zuständen.

c) Zwei minimale DFAs, welche jeweils komplementäre Sprachen akzeptieren, haben gleich viele Zustände.

d) Zwei minimale NFAs, welche jeweils komplementäre Sprachen akzeptieren, haben gleich viele Zustände.

Lösungsvorschlag

a) Richtig: Wie in der Vorlesung gezeigt wurde, gibt es einen NFA mit $n+2$ Zuständen, der die Sprache $(a+b)^*a(a+b)^n$ akzeptiert, gleichzeitig aber jeder DFA wenigstens 2^n Zustände benötigt.

b) Falsch: Wenn ein NFA mit 10 Zuständen eine Sprache akzeptiert, dann hat der zugehörige Potenzmengenautomat höchstens $2^{10} = 1024 < 1217$ Zustände.

c) Richtig: Wenn M ein minimaler Automat für $L = L(M)$ ist, dann erhält man einen Automaten $\overline{M}$ mit $\overline{L} = L(\overline{M})$, indem man Nichtendzustände und Endzustände vertauscht (siehe Vorlesung), wobei die Größe des Automaten sich nicht ändert. Wenn $\overline{M}$ nun nicht minimal wäre, dann existiert $\overline{M}'$ mit $\overline{L} = L(\overline{M}) = L(\overline{M}')$ und $|\overline{M}'| < |\overline{M}|$. Der Komplementärautomat von $\overline{M}'$ wiederum, nennen wir ihn M', hat nun weniger Zustände als M, erkennt aber dennoch dieselbe Sprache. Ein Widerspruch zur Minimalität von M.

d) Falsch: Betrachte $L = \{\epsilon\}$ und $\overline{L} = \Sigma^* - \{\epsilon\}$. Dann enthält $\overline{L}$ alle Wörter, die wenigstens ein Zeichen enthalten. L kann man mit einem NFA erkennen, der nur einen einzigen Zustand, einen Endzustand, aber keine Transitionen enthält. Für $\overline{L}$ benötigt man hingegen auf jeden Fall mindestens zwei Zustände, da ϵ nicht akzeptiert werden darf.

Aufgabe T13

Zeigen Sie mit dem Satz von Myhill–Nerode, dass folgende Sprachen nicht regulär sind. Wenn es sich anbietet, können Sie auch mit Abschlußeigenschaften regulärer Sprachen arbeiten.

1. $L_1 = \{\, a^i b^j c^k \mid i = j \text{ oder } j = k \,\}$
2. $L_2 = \{\, a^i b^j \mid i = j \,\}$
3. L_3, die Menge korrekter Python-Programme
4. $L_4 = \{\, u\$v\$w \mid u, v, w \in \{0,1\}^* \text{ und } \mathrm{bin}(u) + \mathrm{bin}(v) = \mathrm{bin}(w) \,\}$

Hier bezeichnet $\mathrm{bin}(w)$ die Zahl, deren Binärdarstellung w ist. Zum Beispiel ist $\mathrm{bin}(0101) = 5$.

Lösungsvorschlag

1. Sei $N = \{a^i \mid i \geq 0\} \subseteq \Sigma^*$. Seien u_1, u_2 zwei verschiedene Wörter aus N. Somit gilt $u_1 = a^i$, $u_2 = a^j$ und o.B.d.A sei $0 \leq i < j$. Dann ist $w = b^j$ ein trennendes Wort, d.h. $u_1 w = a^i b^j \notin L_1$ aber $u_2 w = a^j b^j \in L_1$. Somit gilt $u_1 \not\equiv_{L_1} u_2$. Es ist also $\text{index}(\equiv_{L_1}) \geq |N| = \infty$. Nach dem Satz von Myhill-Nerode kann L_1 also nicht regulär sein.

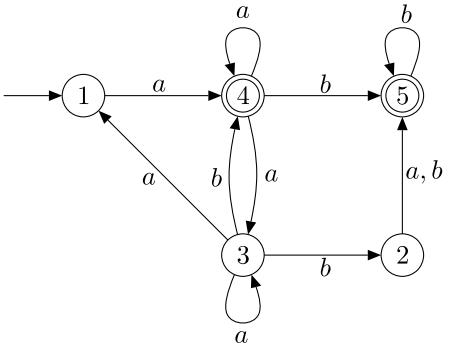
2. Diese Sprache ist nicht regulär. Man kann dafür den selben Beweis nutzen wie im ersten Teil oder aber auch Abschlusseigenschaften nutzen. Dies machen wir nun im Folgenden. Angenommen L_2 sei regulär. Wir definieren uns den Homomorphismus h mit $h(a) = b$ und $h(b) = c$. Dann ist $L_1 = L_2 c^* \cup a^* h(L_2)$. Da reguläre Sprachen unter Vereinigung und Homomorphismus abgeschlossen ist, müsste L_1 regulär sein. Da aber L_1 nicht regulär ist, kann L_2 nicht regulär sein

3. Sei $N = \{(^i \texttt{1} \mid i \geq 0\} \subseteq \Sigma^*$. Seien u_1, u_2 zwei verschiedene Wörter aus N. Somit gilt $u_1 = (^i \texttt{1}$, $u_2 = (^j \texttt{1}$ und o.B.d.A sei $0 \leq i < j$. Dann ist $w =)^j$ ein trennendes Wort, d.h. $u_1 w = (^i \texttt{1})^j \notin L_3$ aber $u_2 w = (^j \texttt{1})^j \in L_3$. Beachten Sie, dass richtig geklammerte Terme wie `(((5)))` ein korrektes Pythonprogramm darstellen, aber falsch geklammerte Terme nicht. Somit gilt $u_1 \not\equiv_{L_3} u_2$. Es ist also $\text{index}(\equiv_{L_3}) \geq |N| = \infty$. Nach dem Satz von Myhill-Nerode kann L_3 also nicht regulär sein.

4. Sei $N = \{0\$v\$ \mid v \in 1^*\{0,1\}^*\} \subseteq \Sigma^*$. Seien u_1, u_2 zwei verschiedene Wörter aus N. Somit gilt $u_1 = 0\$x\$$, $u_2 = 0\$y\$$ für zwei verschieden Wörter x und y. Insbesondere sind $\text{bin}(x)$ und $\text{bin}(y)$ verschieden, da wir führende Nullen verboten haben Dann ist x ein trennendes Wort, d.h. $u_1 x = 0\$x\$x \in L_4$ aber $u_2 x = 0\$y\$x \notin L_4$, da $\text{bin}(0) + \text{bin}(x) = \text{bin}(x)$, aber $\text{bin}(0) + \text{bin}(y) \neq \text{bin}(x)$. Somit gilt $u_1 \not\equiv_{L_4} u_2$. Es ist also $\text{index}(\equiv_{L_4}) \geq |N| = \infty$. Nach dem Satz von Myhill-Nerode kann L_4 also nicht regulär sein.

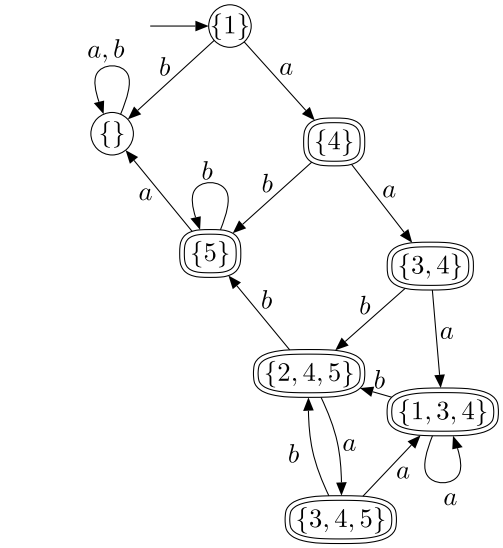
Aufgabe T14

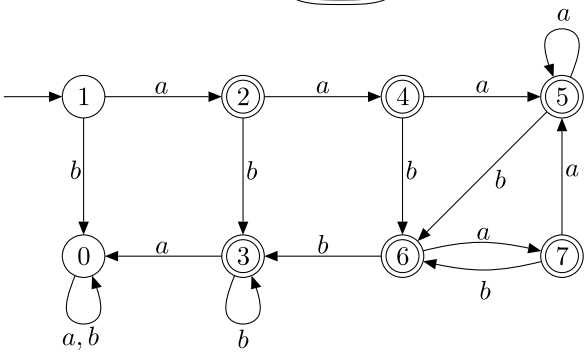
Konstruieren Sie den minimalen deterministischen Automaten zu folgendem NFA. Führen Sie die Potenzmengenkonstruktion aus und geben Sie die Berechnungsschritte der Minimierung an.



Lösungsvorschlag

Wir bilden zuerst den Potenzmengenautomaten und benennen die Zustände der Einfachheit halber um:

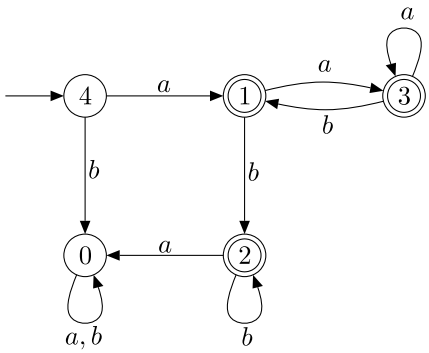




Wir können jetzt den Markierungsalgorithmus aus der Vorlesung anwenden:

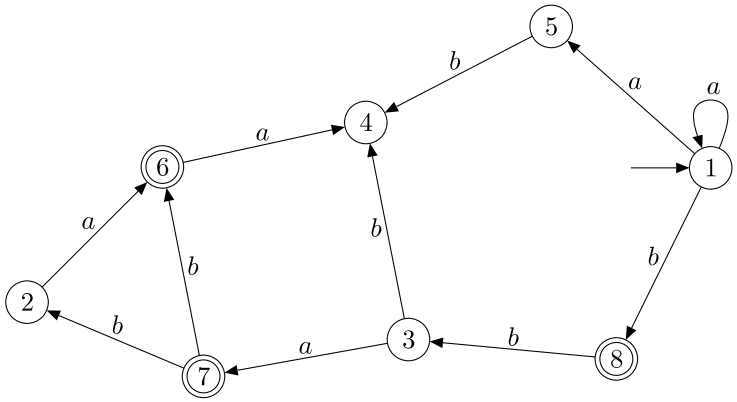
	7	6	5	4	3	2	1
0	X	X	X	X	X	X	X
1	X	X	X	X	X	X	
2	X		X	X	X		
3	X	X	X	X			
4		X					
5		X					
6	X						

Wir sehen, dass die Zustände 4,5 und 7 paarweise äquivalent sind und damit zu einem Zustand zusammengefasst werden können. Das gleiche gilt für das Zustandspaar 2 und 6. Der resultierende Automat sieht wie folgt aus. Die Zustandsbenennung ist hierbei zufällig gewählt und könnte alternativ die jeweils zusammengefassten Zustandsmengen enthalten.



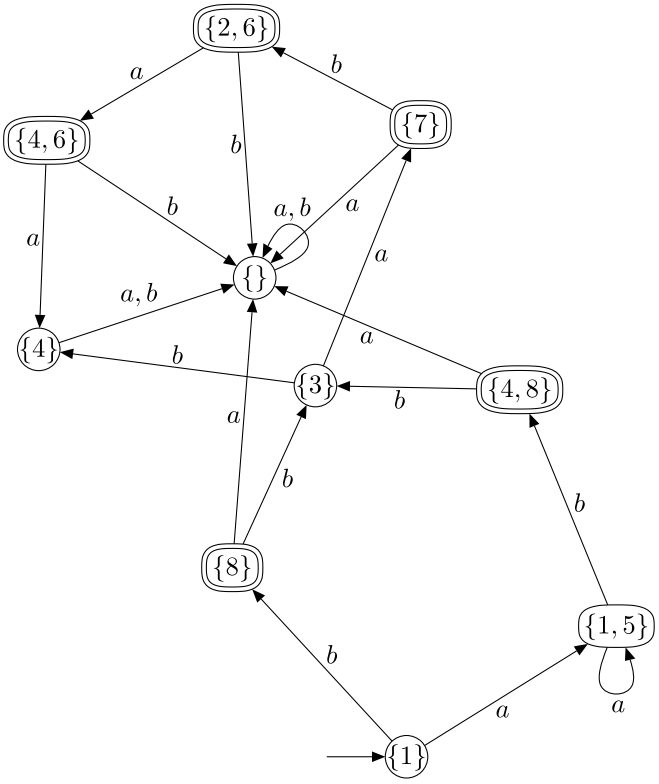
Aufgabe H10 (10 Punkte)

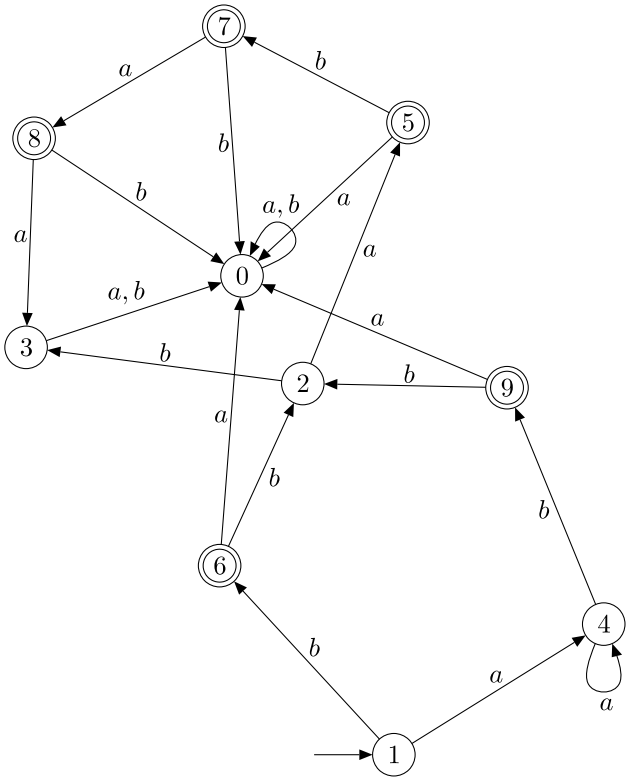
Konstruieren Sie den minimalen deterministischen Automaten zu folgendem NFA. Führen Sie die Potenzmengenkonstruktion aus und geben Sie die Berechnungsschritte der Minimierung an.



Lösungsvorschlag

Wir bilden zuerst den Potenzmengenautomaten und benennen anschließend die Zustände der Einfachheit halber um:

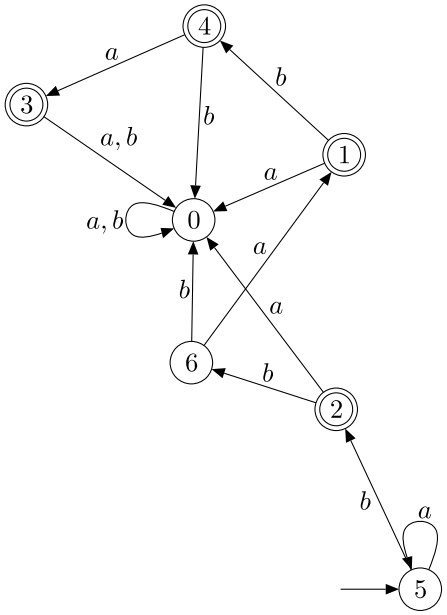




Wir können jetzt den Markierungsalgorithmus aus der Vorlesung anwenden:

	9	8	7	6	5	4	3	2	1
0	X	X	X	X	X	X		X	X
1	X	X	X	X	X		X	X	
2	X	X	X	X	X	X	X		
3	X	X	X	X	X	X			
4	X	X	X	X	X				
5	X	X	X	X					
6		X	X						
7	X	X							
8	X								

Wir sehen, dass die Zustände 1 und 4 äquivalent sind und damit zu einem Zustand zusammengefasst werden können. Das gleiche gilt für die Zustandspaare 0 und 3 sowie 6 und 9. Der resultierende Automat sieht wie folgt aus. Die Zustandsbenennung ist hierbei zufällig gewählt und könnte alternativ die jeweils zusammengefassten Zustandsmengen enthalten.



Aufgabe H11 (10 Punkte)
Welche der folgenden Sprachen sind regulär und welche nicht? Beweisen Sie Ihre Behauptung.

1. $L_1 = \{ a^n \mid \sqrt{n} \in \mathbf{N},\, n > 100 \}$
2. $L_2 = L_1^*$

Die zweite Aufgabe ist nicht einfach und kann nicht unmittelbar mit dem Stoff dieser Vorlesung gelöst werden. Sie brauchen dafür auch Kenntnisse aus anderen Vorlesungen, die Sie wahrscheinlich schon gehört haben. Es ist eine kleine Forschungsaufgabe, durch die Sie sich vielleicht durchbeißen müssen, aber geben Sie nicht auf: Wenn Sie es geschafft haben, steht einer Karriere als Wissenschaftlerin oder Wissenschaftler nichts mehr im Weg.

Lösungsvorschlag

1. Bereits in H8 haben wir gezeigt, dass $L_1' = \{a^{n^2} \mid n \geq 0\}$ nicht regulär ist. Wir beobachten, dass $L_1^{\text{dif}} = L_1' \setminus L_1$ höchstens Wörter der Länge 100 enthält, also endlich und damit regulär ist. Für einen Widerspruch nehmen wir nun an, dass L_1 regulär ist. Dann ist auch $L_1' = L_1 \cup L_1^{\text{dif}}$ regulär, da reguläre Sprachen unter Vereinigung abgeschlossen sind. Ein Widerspruch, und daher ist L_1 nicht regulär sein.

2. Die Sprache ist regulär, da ab einem bestimmten Wortlänge jedes Wort in der Sprache ist. Damit ist das Komplement von L_2 endlich, also ist L_2 regulär.

   Beachte, dass für alle $x \geq 1, y \geq 0$, falls $a^{11^2x+y} \in L_2$, dann auch $a^{11^2(x+1)+y} \in L_2$. Es reicht also zu zeigen, dass für alle $y \in \{1, \ldots, 11^2-1\}$ es $x, z \geq 0$ gibt so dass $y \equiv (x^2)^z \mod 11^2$. Da 23 und $12^2 = 121$ teilerfremd sind, ist $12^2 \equiv 23$ ein Erzeuger der Restklassengruppe $\mathbb{Z}_{121}$. Somit gibt es für alle $y \in \{1, \ldots, 11^2 - 1\}$ ein $z \leq 11^2$ so dass $y \equiv (144)^z \mod 11^2$.

Aufgabe H12 (10 Punkte)
In der Vorlesung haben wir gesehen wie man testen kann, ob ein Wort zur Sprache eines regulären Ausdrucks gehört, der dafür in einen NFA verwandelt wird. Nun betrachten wir konkrete Implementierungen und vergleichen sie.
Es gibt viele Implementierungen, die die Erkenntnisse aus FoSAP ignorieren und direkt auf dem regulären Ausdruck arbeiten. Das sind unter anderem Implementierungen, die es schon seit Jahrzehnten gibt, an denen viel optimiert worden ist und die auch viel genutzt werden. Dazu gehören die Standardbibliotheken vieler Programmiersprachen. In dieser Aufgabe wollen wir herausfinden, ob das eine gute Idee ist.
Aufgabe: Wir betrachten den regulären Ausdruck R in Unixschreibweise `(.*?,){13}z` und Eingabewörter $w_i =$ `1,2,3,4,5,6,7,8,9,10,11,12(,1)`^{i} für $1 \leq i \leq 20$. Zum Beispiel ist $w_3 =$ `1,2,3,4,5,6,7,8,9,10,11,12,1,1,1` . Wir wollen nun herausfinden, ob w_i von R erkannt wird (für $1 \leq i \leq 20$). Dies wird oft als *full match* bezeichnet. Beispielsweise wird `1,1,1,1,1,1,1,1,1,1,1,1,1,z` durch R gematcht.
a) Wählen Sie jeweils mindestens eine Implementierung aus Kategorie 1 und 2 und finden Sie heraus, wie lange es braucht um jedes Eingabewort w_i auf Zugehörigkeit zu R zu testen. Geben Sie ihre Messwerte an und beschreiben Sie, was zu erkennen ist.
b) Woher kommen diese großen Unterschiede in der Laufzeit? Sind Sie überrascht? Formulieren Sie erst eine Vermutung. Recherchieren Sie anschließend und fassen Sie Ergebnisse zusammen. Können Sie einen Fall finden, wo diese Laufzeit sich verheerend in der Realität ausgewirkt hat?
c) Testen Sie nun Ihre NFA-Implementierung von letzter Woche mit derselben Eingabe. Dafür müssen Sie einen NFA erstellen, der R erkennt. Sie dürfen davon ausgehen, dass das Alphabet die alphanumerischen Zeichen und das Komma umfasst. Vergleichen Sie das Resultat mit den

Python	Java
0.0001	0.002
0.0001	0.002
0.001	0.005
0.009	0.010
0.009	0.016
0.009	0.024
0.019	0.064
0.029	0.122
0.049	0.162
0.079	0.111
0.148	0.186
0.277	0.329
0.494	0.589
0.861	1.043
1.524	1.845
3.058	3.211
4.385	5.462
9.512	9.547
16.387	14.871
19.068	23.958

Tabelle 1: Die Laufzeiten von Python und Java auf w_i.

vorherigen. Ist der Vergleich zu den anderen Implementierungen fair? Alternativ dürfen Sie die Implementierung aus der Musterlösung nutzen.

Geben Sie auch Ihren genutzten Quellcode ab!

Kategorie 1: Programme bzw. Standardbibliotheken von Sprachen, die das Problem mit Automaten lösen: egrep, Rust, Go, RE2 von Google.

Kategorie 2: Standardbibliotheken von Sprachen, die das Problem nicht mit Automaten lösen: Java, Python, C++, Perl (vor Version 5.10).

Lösungsvorschlag

a) Man sieht, dass die Laufzeit der Implementierungen aus Kategorie 1 für jede Eingabe sofort das Resultat liefern. Rust ist im 0,2 bis 0,3 Millisekundenbereich, egrep bei 20 bis 30. Bei Kategorie 2 ist die Laufzeit anfangs auch sehr gering, aber sie steigt nach dem Wort Nr. 12 sehr schnell an.

b) Die Implementierungen aus Kategorie 2 haben hier eine sehr langsame Laufzeit, weil es Algorithmen sind, die auf dem regulären Ausdruck *backtracken*. Das bedeutet, dass sie gierig ein Teil des Wortes auf einen Teil des regulären Ausdrucks matchen und wenn dies fehlschlägt, das Matching zum Teil rückgängig machen müssen um dann anders zu matchen. Detaillierte Erklärung gibts es unter Anderem hier: `https://swtch.com/~rsc/regexp/regexp1.html`

   Das liegt daran, dass die meisten Implementierungen nicht nur pure reguläre Ausdrücke implementieren, sondern regular expressions, kompatibel zu Perl sind. Diese sind echt mächtiger als reguläre Sprachen und können nicht mit unseren Automaten simuliert werden. Allerdings sollten die Implementierungen auf Automaten zurückfallen, wenn es sich um pure reguläre Ausdrücke handelt.

Es gab schon mehrere Vorfälle, wo ein langsames Regexmatching Websiten stillgelegt haben:

- Stackoverflow war 34 Minuten unerreichbar und Millionen von Programmierern konnten währenddessen ihrer Arbeit nicht nachgehen. Eine ungültig formatierte Frage mit 20.000 Whitespaces und ein regulärer Ausdruck, der unnötige Whitespaces löscht, haben die Server mit seinem Backtracking-Verhalten überladen. Die einzige Konsequenz, die die Entwickler gezogen haben, ist es den regulären Ausdruck durch eine andere Direktive auszutauschen. Vielleicht gibt es noch einen regulären Ausdruck, der backtrackt. Wer weiß? `https://adtmag.com/blogs/dev-watch/2016/07/stack-overflow-crash.aspx`.
- Ein noch schlimmerer Vorfall: Cloudflare hat einen globalen Ausfall bei ihrem HTTP - Verkehr auch wegen eines regulären Ausdrucks, der backtrackt. Der Artikel geht sehr genau darauf ein, was alles schief ging und erklärt am Ende den für uns relevanten Teil. Sie erklären auch das Backtracking-Verhalten und endliche Automaten im Detail. Die Entwickler bei Cloudflare haben erkannt, dass eine Implementierung mit diesem Verhalten zu gefährlich für ihr Geschäft ist und wechseln die Implementierung zu einer mit linearen Laufzeit, nämlich RE2 oder die von Rust. `https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/`

c) Unsere Implementierung ist im Nanosekundenbereich, was der Performanz der Kategorie 1 entspricht und die der Kategorie 2 massiv schlägt. Dies war auch zu erwarten. Das heißt, dass ihr mit eurem Wissen aus dem zweiten Semester professionelle Software wie die Standardbibliothek von C++ oder Java leicht schlagen könnt.

Der Vergleich ist nicht ganz fair, da die Konstruktion des Automaten (z.B. mit der Thomposon-Konstruktion) entfällt. Dies ist aber vernachlässigbar. Allerdings bauen einige Implementierungen in kompilierenden Sprachen den Automaten schon zur Kompilierungszeit und nicht erst zur Laufzeit, wenn dies möglich ist. Dafür muss der reguläre Ausdruck als konstanter Ausdruck im Quellcode vorhanden sein, wie es auch bei unserem Beispiel der Fall ist.