Lösungen zu den Übungsaufgaben im Buch *Automaten und Sprachen: Theoretische Informatik für die Praxis* von Andreas Müller, ISBN 978-3-662-70145-4 (Softcover), ISBN 978-3-662-70146-1 (eBook), https://link.springer.com/book/10.1007/978-3-662-70146-1. Website zum Buch: https://autospr.ch

# Kapitel 4: Reguläre Operationen und reguläre Ausdrücke

**4.1.** Konstruieren Sie für jede der folgenden Sprachen über dem Alphabet $\Sigma = \{0, 1\}$ einen NEA, ohne den Produktautomaten zu verwenden. Reduzieren Sie diesen für die Teilaufgaben a)-d) auf einen minimalen DEA.

a) $L_1 = \{w \in \Sigma^* \mid w \text{ enthält mindestens zwei 1 nacheinander.}\}$

b) $L_2 = \{w \in \Sigma^* \mid w \text{ ist eine durch drei teilbare Binärzahl.}\}$

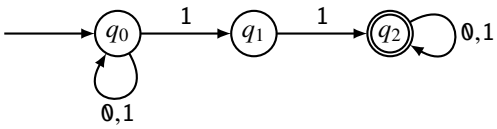
c) $L_3 = L_1 \mid L_2$

d) $L_4 = L_1 L_2$

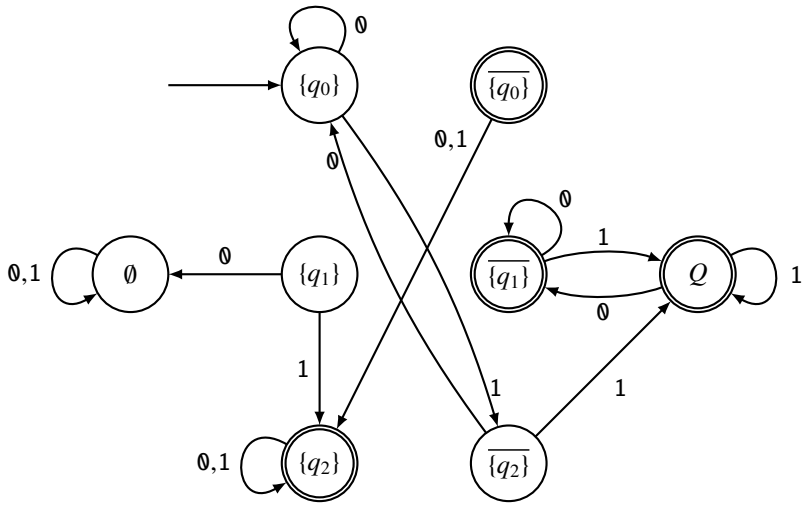
e) $L_5 = L_1^*$

*Hinweis.* Die Teilaufgaben a) und b) waren schon Teil der Aufgabe 3.2 von Kapitel 3.

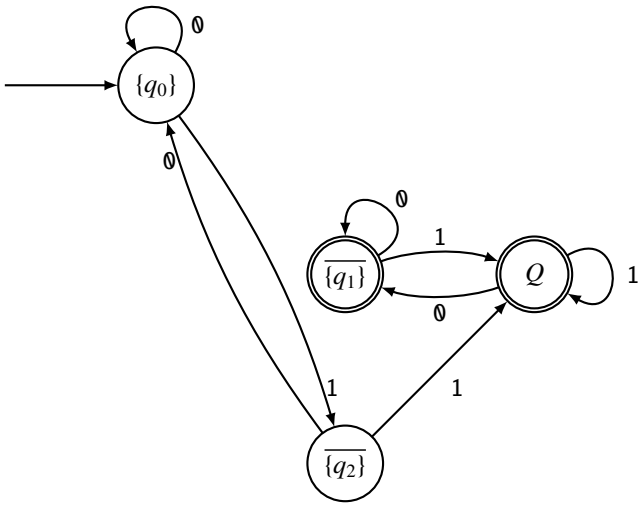
*Lösung.* a) Ein einfacher NEA für $L_1$ ist



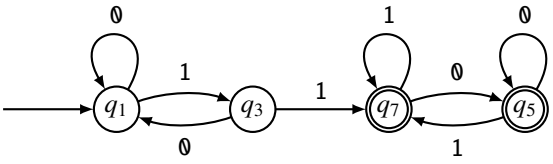

Diesen NEA können wir jetzt mit dem Standardalgorithmus in einen DEA umwandeln:



Eliminieren wir daraus die nicht erreichbaren Zustände, bleibt der DEA




Oder etwas kompakter:



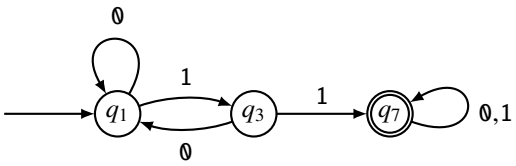

Es ist jedoch noch nicht klar, ob dieser DEA minimal ist. Daher wenden wir den Algorithmus für den Minimalautomaten an, um möglicherweise äquivalente Zustände zu finden. Nach der Markierung von Paaren aus Akzeptierzuständen und anderen Zuständen als nicht äquivalent hat man:

| | $q_1$ | $q_3$ | $q_7$ | $q_5$ |
|---|---|---|---|---|
| $q_1$ | $\equiv$ | | $\times$ | $\times$ |
| $q_3$ | | $\equiv$ | $\times$ | $\times$ |
| $q_7$ | $\times$ | $\times$ | $\equiv$ | |
| $q_5$ | $\times$ | $\times$ | | $\equiv$ |

Im nächsten Schritt bekommt man:

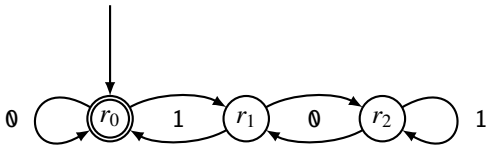
| | $q_1$ | $q_3$ | $q_7$ | $q_5$ |
|---|---|---|---|---|
| $q_1$ | $\equiv$ | $\times$ | $\otimes$ | $\otimes$ |
| $q_3$ | $\times$ | $\equiv$ | $\otimes$ | $\otimes$ |
| $q_7$ | $\otimes$ | $\otimes$ | $\equiv$ | |
| $q_5$ | $\otimes$ | $\otimes$ | | $\equiv$ |

Da das Paar $(q_5, q_7)$ nicht als nicht-äquivalent markiert werden kann, müssen die beiden Zustände äquivalent sein, so dass der minimale Automat $A_1$

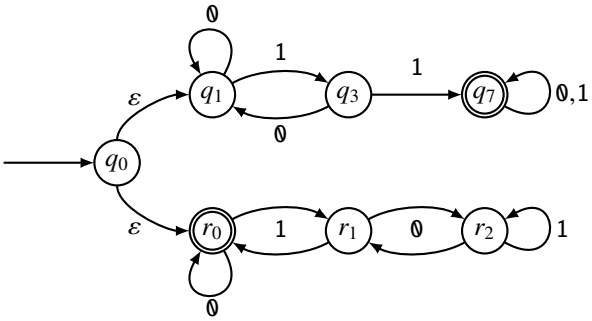



ist.

b) Der Automat $A_2$ für die durch drei teilbaren Binärzahlen wurde in Kapitel 2 entwickelt:




c) Die Alternative $L_1 \mid L_2$ wird durch den NEA



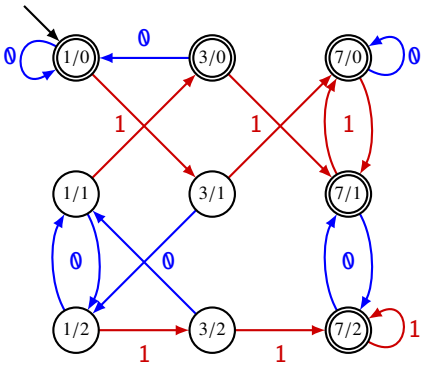

akzeptiert.

Dieser Automat muss jetzt mithilfe des Thompson-NEA in eine DEA umgewandelt werden. Wegen der $\varepsilon$-Übergänge gleich zu Begin ist der Startzustand die Menge

$$\{q_0, q_1, r_1\}.$$

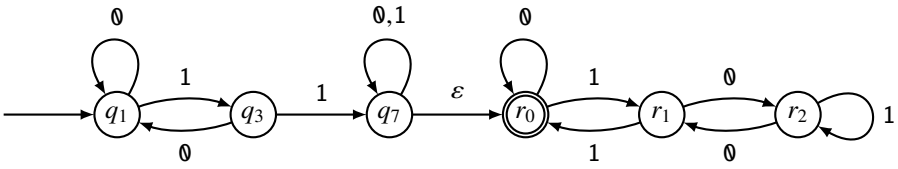
Da es keine Übergängen nach $q_0$ gibt, werden diese $\varepsilon$-Übergänge später nicht mehr verwendet. Da es keine Zeichenüberänge von $q_0$ aus gibt, wird $q_0$ nach dem ersten Übergang verschwinden. Der Zustand $q_0$ spielt also in den Zustandsmengen des Thompson-NEA keine Rolle, wir können ihn daher auch beim Startzustand weglassen.

Alle folgenden Übergänge sind deterministisch, d. h. die Zustandsmengen bestehen immer aus genau einem Zustand das Automaten $A_1$ und einem Zustand des Automaten $A_2$, sie können also in einer $3 \times 3$-Tabelle der Zustandspaare dargestellt werden, genau wie beim Produktautomaten. Beim Übergang werden die Zustände von der jeweiligen Übergangsfunktion abgebildet, wieder wie beim Produktautomaten. Die Konstruktion des Produktautomaten hat sich also auf natürliche Weise wieder ergeben. Der Automat ist

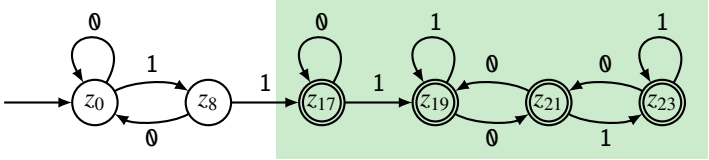



Die Minimierung zeigt wie in Kapitel 3, dass sich die drei Zustände in der dritten Spalte zusammenlegen lassen.

d) $L_4$ ist die Verkettung der zwei Sprachen $L_1$ und $L_2$. Den zugörigen NEA konstruiert man, indem man die beiden Teilautomaten mit einem $\varepsilon$-Übergang aneinanderhängt:




Die ersten drei Zustände bilden einen DEA, in den Zustandsmengen des Thompson-NEA ist also immer genau einer dieser Zustände vorhanden. Die hinteren drei Zustände können aber a priori in jeder Kombination vorkommen. Somit gibt es $3 \cdot 8 = 24$ mögliche Zustände, was für die Zustandsdiagrammdarstellung etwas unübersichtlich ist. Wir wählen daher die Tabellendarstellung für die Übergangsabbildung, sie ist in Tabelle 1 dargestellt. Es bleibt also der deterministische endlicher Automat




(1)

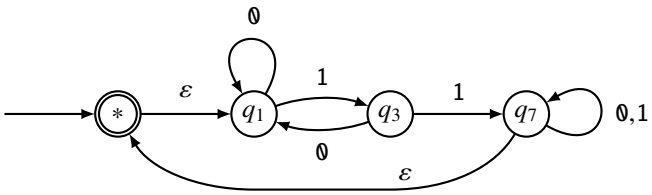
mit sechs Zuständen.

Es ist klar, dass der Automat (1) nicht minimal ist. Es ist aber auch nicht nötig, den Kreuzchenalgorithmus durchzuführen, denn es ist offensichtlich, dass die grün hinterlegten Zustände sich alle zuammenlegen lassen. Der minimierte Automat ist also der, der für die Sprache $L_1$ gefunden wurde.

| Zustand | $q_1$ | $q_3$ | $q_7$ | $r_0$ | $r_1$ | $r_2$ | 0 | 1 |
|---|---|---|---|---|---|---|---|---|
| 0 | ✓ | | | | | | 0 | 8 |
| 1 | ✓ | | | ✓ | | | | |
| 2 | ✓ | | | | ✓ | | | |
| 3 | ✓ | | | ✓ | ✓ | | | |
| 4 | ✓ | | | | | ✓ | | |
| 5 | ✓ | | | ✓ | | ✓ | | |
| 6 | ✓ | | | | ✓ | ✓ | | |
| 7 | ✓ | | | ✓ | ✓ | ✓ | | |
| 8 | | ✓ | | | | | 0 | 17 |
| 9 | | ✓ | | ✓ | | | | |
| 10 | | ✓ | | | ✓ | | | |
| 11 | | ✓ | | ✓ | ✓ | | | |
| 12 | | ✓ | | | | ✓ | | |
| 13 | | ✓ | | ✓ | | ✓ | | |
| 14 | | ✓ | | | ✓ | ✓ | | |
| 15 | | ✓ | | ✓ | ✓ | ✓ | | |
| 16 | | | ✓ | | | | | |
| 17 | | | ✓ | ✓ | | | 17 | 19 |
| 18 | | | ✓ | | ✓ | | | |
| 19 | | | ✓ | ✓ | ✓ | | 21 | 19 |
| 20 | | | ✓ | | | ✓ | | |
| 21 | | | ✓ | ✓ | | ✓ | 19 | 23 |
| 22 | | | ✓ | | ✓ | ✓ | | |
| 23 | | | ✓ | ✓ | ✓ | ✓ | 21 | 23 |

Tabelle 1: Tabelle der Übergangsfunktion des Thompson-NEA. Die nicht erreichbaren Zustände sind grau dargestellt.

e) Die Stern-Konstruktion liefert den folgenden Automaten für $L_5$




○

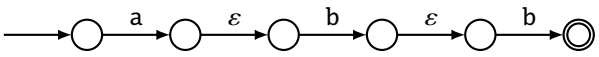
**4.2.** Verwandeln Sie die folgenden regulären Ausdrücke in NEAs über dem Alphabet $\Sigma = \{a, b\}$:

a) `a(abb)*|b`

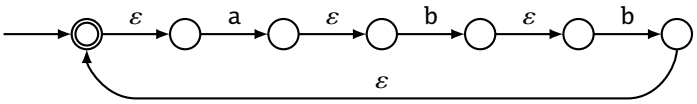
b) `a+|(ab)+`

*Lösung.* Wir wenden die Standardkonstruktionen für die Umwandlung eines regulären Ausdrucks in einen NEA an. Die regulären Ausdrücke werden dabei "von innen nach außen" übersetzt.

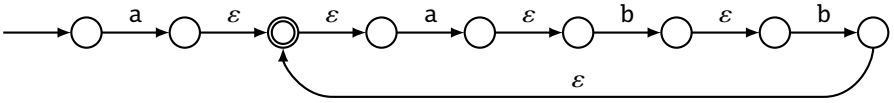
a) Der Teilausdruck `abb` ist eine Verkettung von drei NEAs, die genau ein ein Zeichen langes Wort akzeptieren:



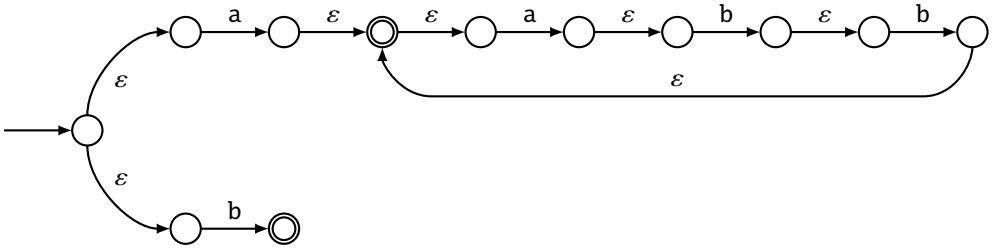

Für die $*$-Operation braucht man einen zusätzlichen Akzeptierzustand und "Rückwärts"-Pfeile:



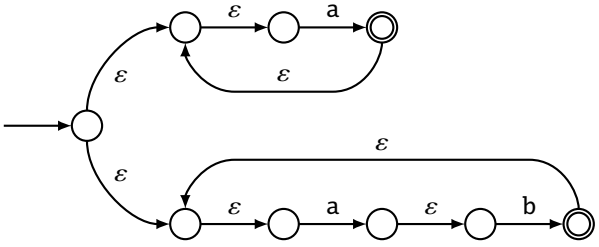

Dem $(\mathtt{abb})^*$ muss jetzt noch ein a vorangehen:



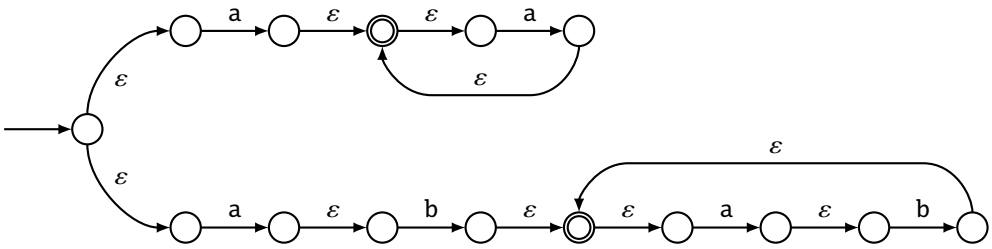

Zum Schluss müssen wir eine Alternative zwischen diesem Ausdruck und dem Ausdruck b bilden:




b) Es stellt sich die Frage, wie man $r^+$ implementieren will. Man könnte $r^+$ als genau wie $r^*$ implementieren, die Konstruktion aber so modifizieren, dass aber das leere Wort doch nicht akzeptiert wird. Dazu würde man den neuen Start-/Akzeptierzustand, der in der Konstruktion von $r^*$ vorkommt, nur als Startzustand, nicht aber als Akzeptierzustand hinzufügen. Dann sähe der Automat wie folgt aus:




Man könnte aber auch $r^+ = rr^*$ implementieren, also als Verkettung eines Automaten, der $r$ akzeptiert mit einer $*$-Konstruktion von $r$. Dann sähe der Automat wie folgt aus:




○

**4.3.** Mitglieder sogenannter Netzgruppen (netgroups) in Unix sind Tripel bestehend aus Hostname, Username und Domain. Es wird die übliche mathematische Notation für Tripel verwendet, zum Beispiel sind

```
( asterix,  hans, )
( obelix ,  heiri, )
( asterix,, ost.ch )
```

solche Tripel, sie werden auch Netgrouptriples genannt. Die Namen können Buchstaben, Ziffern, Punkt und Unterstrich enthalten, die Komponenten können auch leer sein.

a) Ist die Sprache $L_1$ der Netgrouptriples regulär?

b) Zusätzlich wird jetzt verlangt, dass mindestens eine der drei Komponenten nicht leer ist, wir nennen diese neue Sprache $L_2$. ist $L_2$ regulär?

*Lösung.* a) Es gibt einen regulären Ausdruck, der die Wörter der Sprache $L_1$ akzeptiert: also ist die Sprache $L_1$ regulär.

b) Die Sprache $L_\emptyset$ der leeren Netgrouptriples ist regulär, denn sie ist die Sprache der Wörter, die auf den regulären Ausdruck passen, $L_\emptyset = L(\texttt{( *, *, *)})$. Dann ist aber auch $L_2 = L_1 \setminus L_\emptyset$ als Differenz regulärer Sprachen regulär. ○

**4.4.** Ein Wort über dem Alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$ heißt 3-periodisch, wenn es nur aus Wiederholungen der ersten drei Zeichen besteht, also zum Beispiel

$$\texttt{011011011}, \quad \texttt{100100100}, \quad \texttt{101101}, \quad \texttt{111},$$

nicht aber

$$\texttt{00}, \quad \texttt{1001}, \quad \texttt{101100}, \quad \texttt{111000}.$$

Ist die Sprache $L$ der 3-periodischen Wörter regulär?

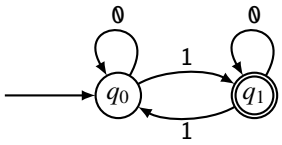
*Lösung.* Ja, denn der reguläre Ausdruck

$$r = \texttt{(000)*|(001)*|(010)*|(011)*|(100)*|(101)*|(110)*|(111)*}$$

akzeptiert die Sprache: $L = L(r)$. ○

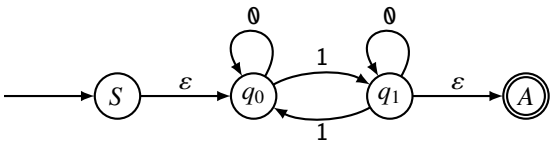
**4.5.** Finden Sie einen regulären Ausdruck für die Sprache $L$ über dem Alphabet $\Sigma = \{0, 1\}$ bestehend aus den Wörtern, die eine ungerade Anzahl 1 enthalten.

*Lösung.* Diese Sprache ist regulär, man kann sofort einen endlichen Automaten dafür angeben:

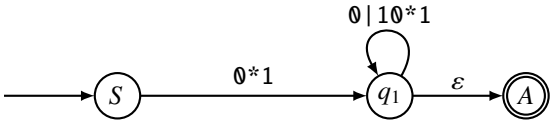



Auf diesen entlichen Automaten kann man jetzt den Standardalgorithmus anwenden, mit dem man einen äquivalenten regulären Ausdruck gewinnen kann.

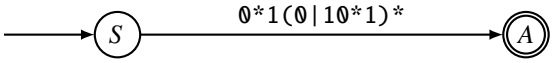
Im ersten Schritt muss man neue Start- und Akzeptierzustände konstruieren:




Im zweiten Schritt müssen die Zwischenzustände entfernt werden, bis nur noch $S$ und $A$ übrig bleiben. Wir beginnen mit $q_0$:




Jetzt kann auch noch $q_1$ entfernt werden:




Der gesuchte reguläre Ausdruck ist also

$$r = \texttt{0*1(0|10*1)*}$$

Eine andere Reihenfolge der Elimination der Zwischenzustände führt zu einem gleichwerten regulären Ausdruck.

Der reguläre Ausdruck besagt etwas salop ausgedrückt, dass zusätzliche 1 immer zu zweien hinzugefügt werden müssen, und dass dabei zwischen den 1 und danach beliebige viele 0 stehen können. Da 1 immer zu zweien hinzugefügt werden, bleibt die Gesamtzahl der 1 unerade. ○

**4.6.** Finden Sie einen regulären Ausdruck sowie einen deterministischen endlichen Automaten für die Sprache $L$ über dem Alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$ bestehend aus Wörtern, in denen der Buchstabe a nie einzeln dasteht, sondern immer nur in Gruppen von mindestens 2 Zeichen a.

*Lösung.* Eine Gruppe von mindestens zwei Zeichen a kann durch den regulären Ausdruck `aaa*` dargestellt werden. Solche Gruppen können jetzt mit beliebig vielen Zeichen b verbunden werden: `(aaa*b*)*`. Dieser Reguläre Ausdruck verlangt aber, dass Wörter mit a beginnen müssen, was nicht den ursprünglichen Forderungen entspricht. Es ist auch erlaubt, dass ein Wort mit beliebig vielen Zeichen b beginnt. Der fertige reguläre Ausdruck wird damit zu

```
b*(aaa*b*)*
```

Ein deterministischer endlicher Automat für die Sprache $L$ ist Selbstverständlich kann man aus diesem endlichen Automaten auch einen regulären Ausdruck gewinnen. Dazu ist zunächst der endlich Automat mit einem neuen Start- und einem einzigen Akzeptierzustand umzuformen: Da keine Pfade über $e$ zu einem Akzeptierzustand führen, kann man in dem VNEA zur Ermittlung des regulären Ausdrucks diesen Zustand weglassen: Jetzt muss man alle Zustände entfernen, wir beginnen mit $q_1$: Jetzt entfernen wir Zustand $q_2$: Und zum Schluss wird jetzt noch $q_0$ entfernt: Der reguläre Ausdruck `(b|aaa*b)*(|aaa*)` ist also auch eine mögliche Lösung der Aufgabe.

Die ursprüngliche Formulierung verlangte, dass a in Gruppen von mindestens zwei Zeichen auftreten musste, was man auch so verstehen konnte, dass einzig das alleinstehende Zeichen a nicht zulässig ist. Diese Sprache ist natürlich auch regulär.

$L = \Sigma^* \setminus \{\mathtt{a}\}$. Da die Sprache {a} endlich ist, ist sie regulär. $\Sigma^*$ ist auch regulär. Die Differenz regulärer Sprachen ist ebenfalls regulär. Man muss also nur noch einen endlichen Automaten und einen regulären Ausdruck dafür finden.

Der reguläre Ausdruck muss ausdrücken, dass ein mit a beginnendes Wort noch mindestens ein Zeichen haben muss. Für Wörter, die mit b beginnen, gibt es keine weiteren Bedingungen. Also:

```
a.+|b.*
```

Auf dieser Basis lässt sich auch leicht ein endlicher Automat formulieren ○

**4.7.** In der Astronomie werden Punkte am Himmel mithilfe der Winkel der geographischen Länge und Breite angegeben, die im astronomischen Zusammenhang auch Rektaszension und Deklination genannt werden. Allerdings sind viele verschiedene Formate für die Winkelangabe gebräuchlich, manchmal werden sogar im gleichen Sternkatalog verschiedene Formate verwendet. Astronomen haben eine gute Intuition für einen Winkel in Minuten, da der Vollmonddurchmesser ziemlich genau 30 Winkelminuten beträgt. Daher werden Formate, die Winkelminuten zeigen, Dezimalbrüchen vorgezogen. Übliche Formate sind

| Format | Beispiel |
|---|---|
| dezimale Grade | `65.4321` |
| Grad, dezimale Minuten | `65 25.926` |
| Grad, Minuten, dezimale Sekunden | `65 25 55.56` |

Natürlich kann der Nachkommateil auch fehlen, und es ist auch möglich, dass ein Winkel negativ ist. Stellen Sie einen regulären Ausdruck auf, der genau diese Formate akzeptiert.

*Lösung.* Eine Winkelangabe endet immer mit einem optionalen Nachkommateil

```
(\.[0-9]*)?
```

Davor stehen maximal drei Gruppen von Ziffern, die jeweils durch Leerzeichen getrennt sind.

```
[0-9]+( *[0-9]+){0,2}
```

Zusammengesetzt, und mit einem optionalen Vorzeichen versehen:

```
(|-|\+)[0-9]+( *[0-9]+){0,2}(\.[0-9]*)?
```

Für die Praxis wäre aber eine andere Form nützlicher, nämlich ein regul"rare Ausdruck, der nicht nur die Gültigkeit des Formates erkennt, sondern auch die einzelnen Teile identifiziert, damit man diese Teilstrings gleich zur Berechnung des Winkelwertes heranziehen kann. Dieser Ausdruck beginnt immer mit einer Gruppe von Ziffern

```
[0-9]+
```

Diese Gruppe kann jetzt entweder von einem Nachkommateil gefolgt sein oder von einer weiteren Gruppe

```
[0-9]+(\.[0-9]*| weitere Gruppen )?
```

Die weiteren Gruppen haben aber genau das gleiche Format wie das, was der bisherige Ausdruck darstellt, man kann also den Ausdruck einfach rekursiv nochmals einpacken

```
(|-|\+)[0-9]+(\.[0-9]*| [0-9]+(\.[0-9]*| [0-9]+(\.[0-9]*)?)?)?
```

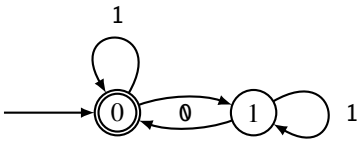
Bei der letzten Gruppe kann nur ein Kommateil folgen, etwas kleineres als Sekunden gibt es ja nicht. ○

**4.8.** Finden Sie einen regulären Ausdruck für die Sprache

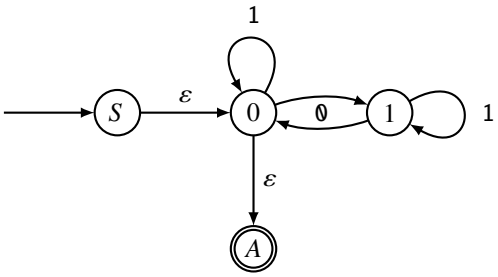
$$L = \{w \in \Sigma^* \mid |w|_0 \text{ ist gerade.}\}$$

über dem Alphabet $\Sigma = \{0, 1\}$.

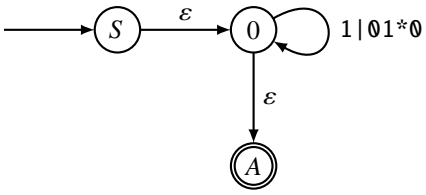
*Lösung.* Wir konstruieren zunächst einen DEA für die Sprache $L$:



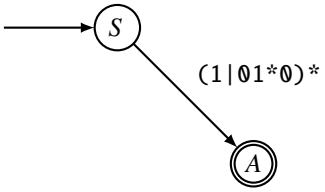

Diesen Automaten müssen wir nun mit separaten Start- und Akzeptierzuständen ausstatten:




Jetzt entfernen wir nacheinander die Zwischenzustände, beginnend mit dem Zustand 1:

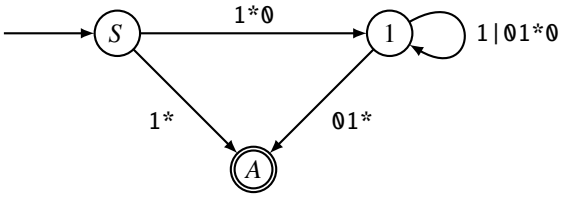



gefolgt vom Zustand 0:

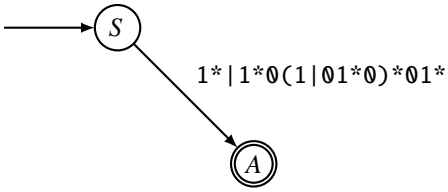



Der reguläre Ausdruck besagt, dass ein Wort in $L$ aus Teilen besteht, die entweder nur aus Einsen bestehen, oder aus Paaren von Nullen, zwischen denen beliebige viel Einsen eingeschoben sein können.

Beginnt man mit dem Enfernen von Zwischenzuständen beim Zustand 0, wird der reguläre Ausdruck ganz anders aussehen:




Jetzt entfernt man den Zustand 1 und erhält




Auch dieser reguläre Ausdruck besagt, dass man zusätzliche Nullen immer nur paarweise einschieben kann, und dass dazwischen jeweils Einsen stehen können. ○

**4.9.** IPv6-Adressen bestehen aus acht Gruppen von jeweils 16 Bits. Eine Gruppe wird als maximal vier Hex-Ziffern `0` – `9`, `a` – `f` codiert, führende Nullen sind in jeder Gruppe erlaubt. Die Gruppen werden durch Doppelpunkt `:` getrennt. Eine typische IPv6-Adresse ist

```
2001:db8:85a3:0:0:8a2e:370:7334
```

Folgen mehr als eine Gruppe aus lauter Nullen aufeinander, im obigen Beispiel die Zeichenfolge `:0:0:`, können diese durch einen doppelten Doppelpunkt `::` abgekürzt werden, aber nur einmal, da

zum Beispiel in der Adresse `1::1::1` nicht klar wäre, wie lange die Null-Gruppen sind und damit an welcher Position in der Adresse die mittlere 1 eigentlich steht. Die obige Adresse kann also zu

`2001:db8:85a3::8a2e:370:7334`

abgekürzt werden.

Finden Sie einen regulären Ausdruck, mit dem Eingabefelder für IPv6-Adressen plausibilisiert werden können.

*Lösung.* Die einzelnen Gruppen werden durch den regulären Ausdruck

$$r_1 = \texttt{[0-9a-f]\{1,4\}}$$

beschrieben. Dieser Ausdruck akzeptiert das leere Wort nicht.

Eine typische IPv6-Adresse ohne die Abkürzung von Nullgruppen wird durch den regulären Ausdruck

$$\begin{aligned} r_2 &= r_1\texttt{(:}r_1\texttt{)\{7\}} \\ &= \texttt{[0-9a-f]\{1,4\}(:[0-9a-f]\{1,4\})\{7\}} \end{aligned}$$

beschrieben.

Wenn nur mindestens zwei Nullgruppen abgekürzt werden sollen, dann besteht eine IPv6-Adresse aus zwei Teiladressen, die möglicherweise leer sind, getrennt von einem doppelten Doppelpunkt `::`.

$$\begin{aligned} & r_3 = r_4\texttt{::}r_4 \\ \text{mit} \quad & r_4 = \texttt{(}r_1\texttt{(:}r_1\texttt{)\{,6\})?} \\ & \quad = \texttt{([0-9a-f]\{1,4\}(:[0-9a-f]\{1,4\})\{,6\})?} \\ & \quad = \texttt{([0-9a-f]\{1,4\}(:[0-9a-f]\{1,4\})\{,6\})?} \end{aligned}$$

Eine beliebige IPv6-Adresse passt daher auf den regulären Ausruck

$$r = r_2 \mid r_3,$$

der jedoch zu lange ist, ihn auf einer einzigen Zeile darzustellen.

Dieser Ausdruck stellt nicht sicher, dass die Gesamtzahl von Gruppen auf beiden Seiten des doppelten Doppelpunktes nicht größer als 6 ist. Dazu müssten man zusätzliche Ausrücke ähnlich wie $r_4\texttt{::}r_4$ bauen, jedoch $r_4$ ersetzen durch Ausdrücke, die die Anzahl der Gruppen limitiert. Dies scheint jedoch übermäßig kompliziert und wird daher sinnvoller der Anwendungslogik überlassen. ○