Reguläre Sprachen / Typ 3

Hans Jürgen Ohlbach

Keywords: Typ 3-Sprachen, endliche Automaten, Wortproblem, Parsing von Typ 3-Sprachen, Pumping Lemma

Empfohlene Vorkenntnisse: Funktionen und Relationen, Formale Sprachen

Inhaltsverzeichnis

1 Einführung

In der Chomsky Hierarchie der formalen Sprachen gibt es als speziellsten Typ den Typ 3, genannt auch, die *regulären Sprachen*. Sie werden durch eine Typ 3-Grammatik erzeugt. Bei Typ 3-Grammatiken über einem Alphabet Σ haben die Produktionsregeln die Form

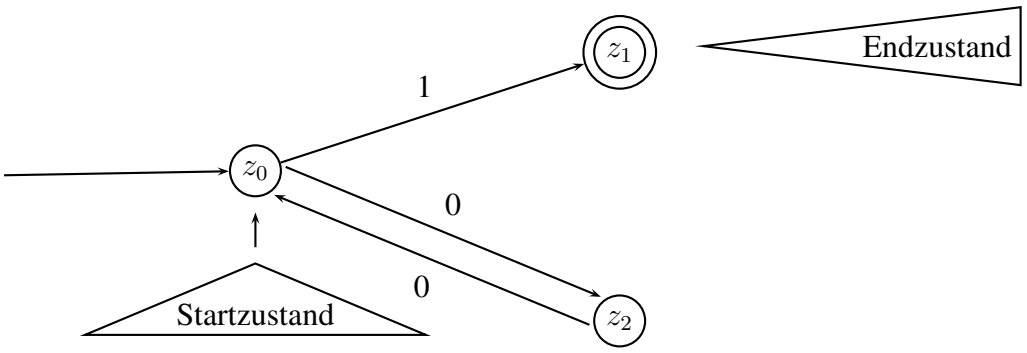
$$A \to aB$$
$$A \to a$$

wobei A, B Variablen (Nicht-Terminalsymbole), und $a \in \Sigma$ eine Konstante (Terminalsymbol) ist.

2 Endliche Automaten

Das *Wortproblem* für eine formale Sprachen L über einem Alphabet Σ besteht darin, herauszufinden, ob ein gegebenes Wort $w \in \Sigma^*$ in der Sprache L ist, oder nicht. Den Vorgang, dieses für ein gegebenes Wort herauszufinden, bezeichnet man auch als *Parsing*. Eine der Mechanismen, mit denen man das herausfinden kann, sind *endliche Automaten*

Ein Automat besteht aus *Zuständen* und *Übergängen*. Ein Beispiel für einen solchen Automaten ist:

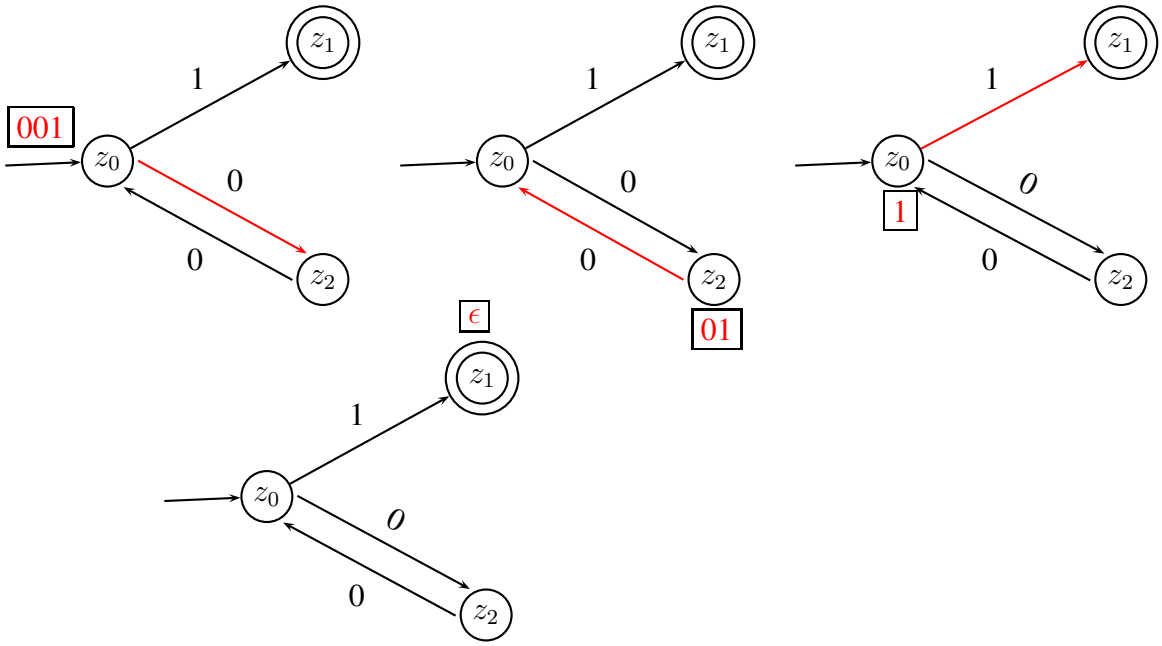


z_0, z_1, z_2 sind die *Zustände*, und die Pfeile sind die Übergänge. Die Übergänge sind mit Buchstaben aus dem Alphabet Σ gelabelled. Es gibt *einen* Startzustand und einen oder mehrere Endzustände (mit Doppelkreis).

Der Automat auf dem Bild oben ist in der Lage, die Sprache $L = (00)^+1$ zu parsen. L enthält die Wörter, die mit mindestens einer Doppelnull starten, und am Ende eine 1 haben, also z.B. 001, oder 00001, oder 0000001 usw.

Um ein solches Wort zu parsen startet der Automat in dem Startzustand mit dem Wort als Eingabe. In jedem Zustand Z, beginnend mit dem Startzustand, testet er, ob es für das führende Zeichen in dem Wort einen Übergang zu einem Zustand Z' mit diesem Zeichen als Label gibt. Wenn ja, löscht er das führende Zeichen aus dem Wort und geht in den Zustand Z'. Wenn nein, endet der Automat mit: *Wort ist nicht akzeptiert*. Wenn der Automat in einen Endzustand gekommen ist, und gleichzeitig das Wort leer geworden ist, dann endet der Automat mit: *das Wort ist akzeptiert.*

Die folgenden Bilder zeigen die Arbeitsweise des Automaten beim parsen des Wortes 001.



Das Wort ist leer geworden, und der Automat endet in einem Endzustand.

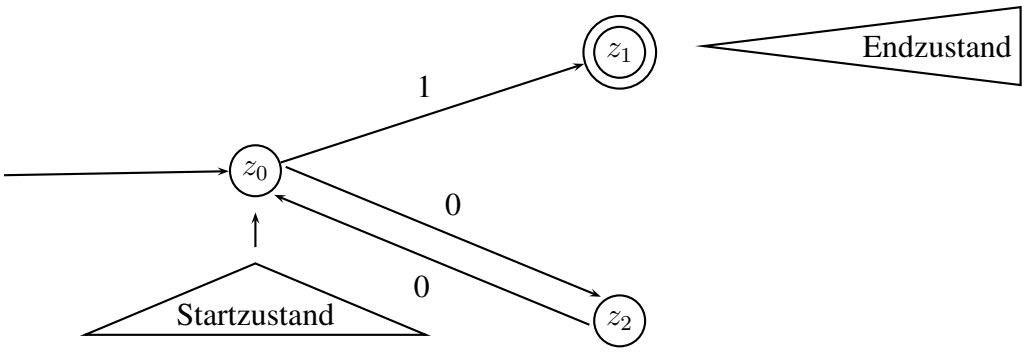
Die formale Definition eines endlichen Automaten ist:

Definition 1 (Deterministicher Endlicher Automat (DEA))
Ein deterministischer endlicher Automat $M = (Z, \Sigma, \delta, z_0, E)$ besteht aus:

- *einer endlichen Menge Z von Zuständen*
- *einem Eingabealphabet Σ*
- *einem Startzustand z_0*
- *einer Menge E von Endzuständen*
- *einer Überführungsfunktion $\delta : Z \times \Sigma \to Z$, die für jeden Zustand und jede Eingabe festlegt, welcher der Nachfolgezustand ist.*
  Wir schreiben $z \overset{a}{\mapsto} z'$ falls $\delta(z, a) = z'$

Die Abkürzung für Deterministischer Endlicher Automat ist: DEA oder DFA (Deterministic Finite Automaton)

Das Bild von oben zeigt einen solchen Automaten:



Er ist modelliert allerdings nicht exakt, was in der Definition gefordert wird. Z.B. gibt es von z_2 keinen Übergang für den Buchstaben 1. Die zugehörige Übergangsfunktion δ ist daher dafür nicht definiert. Hierfür bieten sich zwei Auswege an. Der einfachere Ausweg, ist, dass die Übergangsfunktion δ *partiell* sein darf, d.h. sie muss nicht überall definiert sein. Der zweite Ausweg ist, dass man einen *Sackgassenzustand* einführt, in den die fehlenden Übergänge hineinführen, aber kein Übergang mehr herausführt.

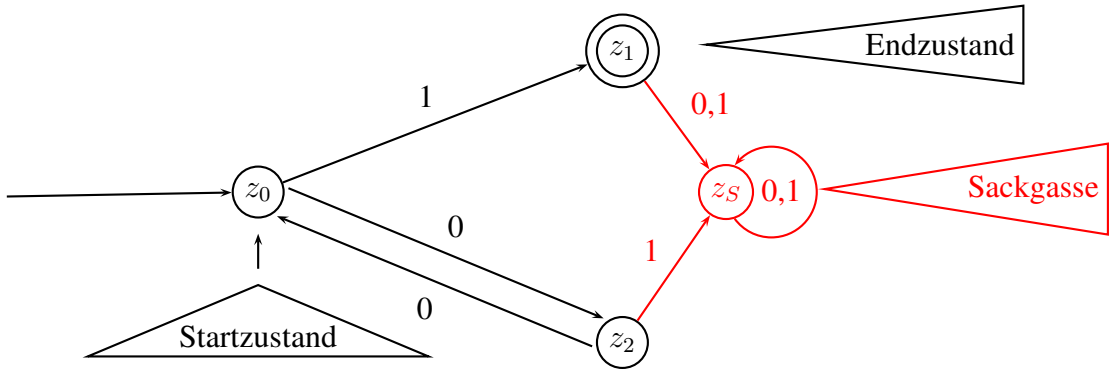
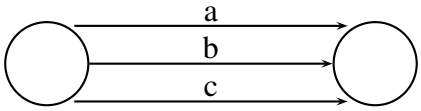


Abbildung 1: DEA für (00)*1

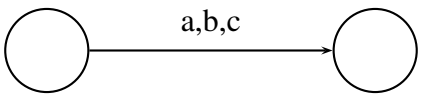
Für einige theoretische Zwecke ist es praktischer, die Übergangsfunktion als total anzunehmen, wobei man einen solchen Sackgassenzustand trotzdem einfach weglässt.

In dem obigen Bild wurde eine abkürzende Notation benutzt:

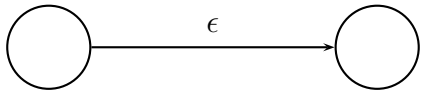
Anstelle von



zeichnet man



Manchmal ist auch ein sog. ϵ-Übergang nützlich. Er beschreibt einen Übergang in einen anderen Zustand, ohne Einfluss auf das zu parsende Wort.



Dafür erweitert man die Übergangsfunktion entsprechend: $\delta(z, \epsilon) = z'$

Die soeben informell eingeführte Arbeitsweise eines DEA kann man auch formal definieren:

Definition 2 (Arbeitsweise eines DEA)
Eine DEA $M = (Z, \Sigma, \delta, z_0, E)$ kann folgendermaßen benutzt werden, um ein Wort $w \in \Sigma^*$ zu untersuchen:
Der Automat startet mit w im Startzustand z_0. Ist der Automat in einem Zustand z_i mit Restwort v macht er folgendes:

- *falls z_i ein Endzustand ist, und v leer geworden, dann terminiert er mit „akzeptiert“,*
- *falls $v = av'$ und $\delta(z_i, a) = z_j$, dann geht er mit dem Restwort v' in den Zustand z_j,*
- *für $\delta(z_i, \epsilon) = z_j$, dann geht er mit dem Wort v in den Zustand z_j,*
- *sobald das Wort leer geworden ist, aber der Automat nicht in einem Endzustand ist, dann terminiert er mit „nicht akzeptiert“.*

Für den DEA M definieren wir als $L(M)$ die Menge der Wörter, die M akzeptiert:

$$L(M) = \{w \in \Sigma^* \mid M \text{ terminiert bei Eingabe von } w \text{ mit „akzeptiert“}\}$$

Der Automat ist *deterministisch*, weil es in jedem Zustand *genau einen* Nachfolgezustand gibt.

Ein solcher Automat akzeptiert also einige Wörter, und andere Wörter akzeptiert er nicht. Wir werden sehen, dass es genau die Wörter der Typ 3-Sprachen sind, die er akzeptiert.

Eine weiteres Beispiel für einen DEA ist der folgende. Er akzeptiert alle Bitfolgen, die irgendwo 00 enthalten.

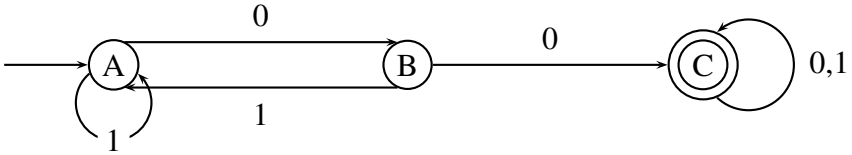
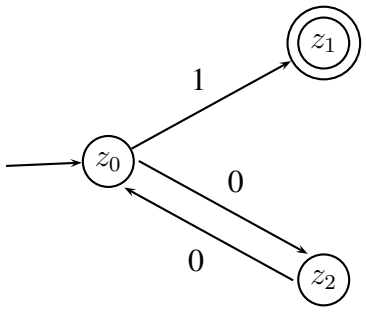


Abbildung 2: DEA für {0,1}*00{0,1}*

Jeder DEA kann in eine reguläre Grammatik übersetzt werden: Die Zustände des DEA werden zu Variablen der Grammatik, und die Übergänge werden zu den Produktionen. Der Startzustand des DEA wird zur Startvariablen.

Definition 3 (DEA $\mapsto$ reguläre Grammatik)
Ein DEA $M = (Z, \Sigma, \delta, z_0, E)$ wird in eine reguläre Grammatik $G = (Z, \Sigma, P, z_0)$ mit $P = \{z \to x\delta(Z, x) \mid z \in Z, x \in \Sigma\} \cup \{z \to x \mid z \in Z, \delta(Z, x) \in E, x \in \Sigma\}$ übersetzt.

Mit dieser Übersetzung wird der DEA aus Abbildung 2:



zu:

$z_0 \to 1z_1$
$z_0 \to 0z_2$
$z_2 \to 0z_0$
$z_0 \to 1$

wobei die erste Produktion überflüssig ist, da es von z_1 nicht weitergeht.

Der DEA aus Abbildung 2 wird zu:

A $\to$ 0B	A $\to$ 1A	B $\to$ 1A
B $\to$ 0C	C $\to$ 0C	C $\to$ 1C
B $\to$ 0	C $\to$ 0	C $\to$ 1

Theorem 1 *Falls ein Wort w von einem DEA M akzeptiert wird, dann wird w von der aus M übersetzten Grammatik produziert.*

Man kann den Durchlauf durch den Automaten für das Wort $w = a_1 \dots a_{n-1}$:
$S_1 \overset{a_1}{\mapsto} S_2 \dots S_{n-1} \overset{a_{n-1}}{\mapsto} S_n$
vom Startknoten S_1 bis zum Endknoten S_n direkt in eine Ersetzungssequenz der Grammatik:
$S_1 \mapsto a_1 S_2, \dots, S_{n-1} \mapsto a_{n-1}$
übersetzen, so dass aus der Startvariablen das Wort w erzeugt wird.

3 Nichtdeterministische Automaten

Wir möchten natürlich auch die Umkehrung haben: Reguläre Grammatik $\mapsto$ DEA.

Hier gibt es jedoch eine zunächst massive Schwierigkeit: Eine Grammatik kann Produktionen der Art $S \to aA, S \to aB$ haben. D.h. für *den gleichen* Buchstaben kann es mehrere Produktionen geben. Das führt in dem zugehörigen Automaten dazu, dass man von einem Zustand aus mit dem gleichen Buchstaben unterschiedliche Zustände erreichen kann. Damit wird der Automat *nichtdeterministisch*.

Definition 4 (Nichtdeterministicher Endlicher Automat (NEA,NFA))
Ein nichtdeterministischer endlicher Automat $M = (Z, \Sigma, \delta, S, E)$ besteht aus:

- *einer endlichen Menge Z von Zuständen*
- *einem Eingabealphabet Σ*
- *einer Menge S von Startzuständen*
- *einer Menge E von Endzuständen*
- *einer Überführungsfunktion $\delta : Z \times \Sigma \to \mathcal{P}(Z)$, die für jeden Zustand und jede Buchstaben eine* Menge *von Nachfolgezuständen festlegt.*

Der NEA unterscheidet sich also vom DEA, zum einen, dass er mehrere Startzustände haben kann, und zum anderen, dass es von einem Zustand und einem Buchstaben aus mehrere Nachfolgezustände geben kann.

Der folgende Automat ist nichtdeterministische, weil es von Zustand B für den Buchstaben b einen Übergang nach A und einen Übergang nach C gibt. Er akzeptiert die Sprache $\{ab, aba\}^*$

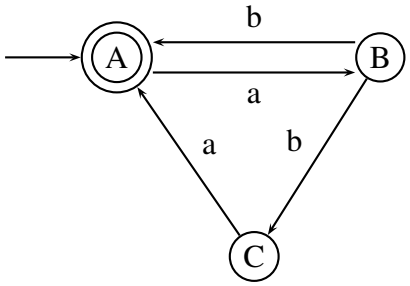


Abbildung 3: Automat für $\{ab, aba\}^*$.

Der Automat enthält zwei Zyklen: A$\mapsto$B$\mapsto$A und A$\mapsto$B$\mapsto$C$\mapsto$A, die beliebig ineinander verschachtelt werden können.

Um ein Wort zu parsen müssen nichtdeterministische Automaten also bei einem Zustand, bei dem es mehrere Alternativen gibt, eine Alternative *raten*. Falls die falsche Alternative geraten wurde, und

das Wort dann nicht akzeptiert wird, muss der Automat zum letzten Ratepunkt zurückgehen (Backtracking), und die nächste Alternative ausprobieren.

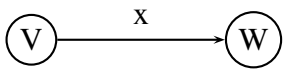
Ein Wort ist dann akzeptiert wenn immer die richtige Alternative gefunden wurde, so dass der Automat dann wie bei einem DEA das Wort akzeptiert. Da es auch mehrere Startzustände geben kann, muss auch der richtige Startzustand gefunden werden. Das Wort muss also keinesfalls für alle Alternativen akzeptiert werden. Es genügt, bei jeder Verzweigung eine passende zu finden.

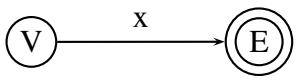
Das ist natürlich sehr aufwendig, und, zum Glück, auch nicht notwendig. Man kann nämlich jeden nichtdeterministischen Automaten in einen deterministischen Automaten transformieren.

Für einen NEA M definieren wir die Menge der akzeptierten Wörter zu:

$$L(M) = \{w = a_1 \dots a_{n-1} \in \Sigma^* \mid \textit{es gibt für } w \textit{ einen akzeptierenden Durchlauf } S_1 \overset{a_1}{\mapsto} S_2 \dots S_{n-1} \overset{a_{n-1}}{\mapsto} S_n \text{ durch den Automaten}\}$$

Jetzt können wir eine reguläre Grammatik in einen NEA übersetzen. Dabei nutzt man aus, dass die regulären Produktionen entweder die Gestalt $V \to xW$ oder die Gestalt $V \to x$ haben.

$V \to xW$ wird übersetzt in einen Übergang 

$V \to x$ wird übersetzt in einen Übergang 

wobei E ein zusätzlicher Zustand ist, der als Endzustand fungiert.

Definition 5 (Reguläre Grammatik $\mapsto$ NEA)
Eine reguläre Grammatik $G = (V, \Sigma, P, S)$ wird in einen NEA $M = (V \cup \{E\}, \Sigma, \delta, S, \{E\})$ übersetzt, mit der Übergangsfunktion δ:

$$\delta(V, x) = \begin{cases} \{E\} & \textit{falls } `V \to x' \in P \\ \{W \mid `V \to xW' \in P\} & \textit{sonst} \end{cases}$$

Auf diese Weise kann man die obigen Übersetzungsbeispiele DEA $\mapsto$ Grammatik wieder zurückübersetzen (Die Benennung der Zustände ist natürlich beliebig).

Theorem 2 *Für eine reguläre Grammatik G gilt: Jedes Wort $w \in L(G)$ wird von dem aus G generierten NEA akzeptiert.*

Der Beweis dreht die Argumentation von Theorem 1 um: Für $w = a_1 \dots a_{n-1} \in L(G)$ gibt es einen (linear degenerierten) Syntaxbaum, der aus der Folge von Ersetzungen besteht:
$S_1 \to a_1 S_2, \dots, S_{n-1} \to a_{n-1}$
Diese lässt sich direkt in eine Folge
$S_1 \overset{a_1}{\mapsto} S_2 \dots S_{n-1} \overset{a_{n-1}}{\mapsto} S_n$
von Übergängen des Automaten vom Startknoten S_1 bis zum Endknoten S_n übersetzen. Also akzeptiert der Automat das Wort.

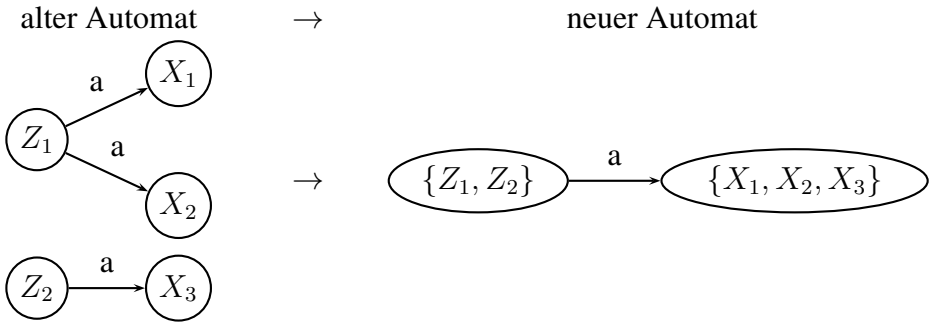
Zusammen mit Theorem 1 bedeutet das, dass reguläre Sprachen genau diejenigen sind, deren Wörter von endlichen Automaten akzeptiert werden

3.1 Nichtdeterministische Automaten deterministisch machen

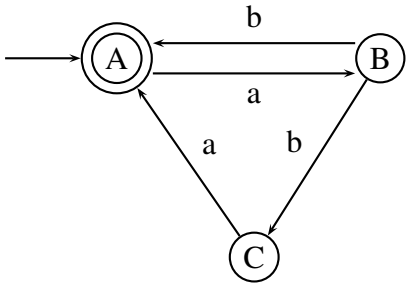
Zum Glück kann man jeden nichtdeterministischen endlichen Automaten auf relativ einfache Weise deterministisch machen.

Dazu fasst man zunächst alle Startzustände zu einem neuen Startzustand zusammen, mit der *Menge* der Zustandsnamen als neuen Zustandsnamen. Von da aus fasst man weitere Zustände zusammen: Für einen Zustand $\{Z_1, \ldots, Z_n\}$ des neuen Automaten, und einen Buchstaben a, fasst man alle Zustände im alten Automaten zusammen, die von einem der Z_i einen a-Übergang haben, und erzeugt dafür einen a-Übergang im neuen Automaten. Die neuen Endzustände $\{E_1, \ldots, E_n\}$ sind alle Zustände, bei denen im alten Automaten einer der E_i ein Endzustand war.

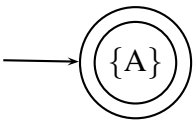
Schematisch könnte das so aussehen.



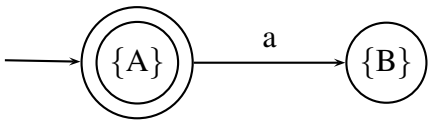
Beispiel: der Automat aus Abbildung 3 für die Sprache $\{ab, aba\}*$ ist nicht-deterministisch.



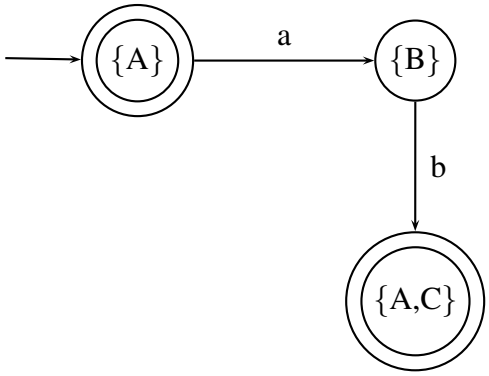
Wir wandeln den schrittweise in einen deterministischen Automaten um. Es gab nur einen Startzustand A. Mit diesem fangen wir den neuen Automaten an.



Von A aus gibt es nur einen einzigen a-Übergang, nach B:

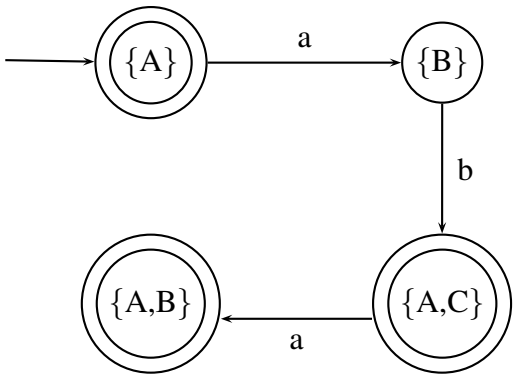


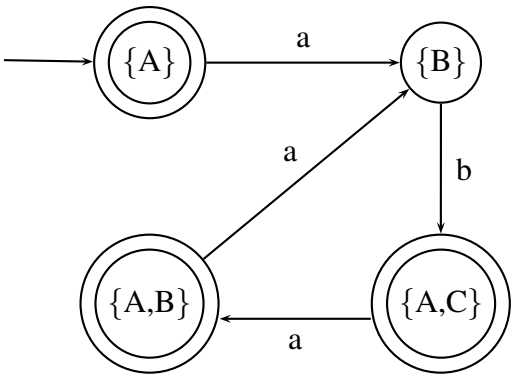
Von B aus gibt es b-Übergänge nach A und C. A ist Endzustand, deshalb wird {AC} ebenfalls ein Endzustand.



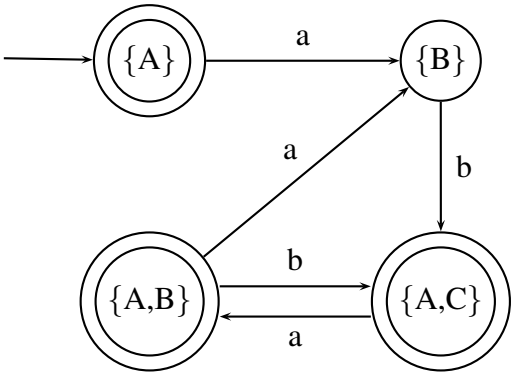
Von A gibt es einen a-Übergang nach B, und von C gibt es einen a-Übergang nach A (linkes Bild). A ist Endzustand, deshalb wird {AB} ebenfalls ein Endzustand.

Von A gibt es dann einen a-Übergang nach B (rechtes Bild):





Von B gibt es einen b-Übergang nach A und C.



Dies ist der endgültige Automat. Er ist deterministisch, und, wie man sich überzeugen kann, er akzeptiert auch die Sprache {ab,aba}*.

Dieses sehr informelle Vorgehen, welches mit Papier und Bleistift gut funktioniert, kann man systematisieren, so dass man es auch implementieren kann. Dazu merkt man sich die Übergänge in einer Matrix, die man systematisch aufbaut. Die Zeilen stehen für die Zustände im neuen Automaten. Für jeden neuen Zustand notieren wir in den Spalten für jeden Buchstaben die Übergänge.

Wir illustrieren das an dem gleichen Beispiel wie oben. Dazu notieren wir uns zunächst die Übergänge im Ausgangsautomaten als Matrix:

δ	a	b
A	B	
B		A,C
C	A	

Jetzt konstruieren wir die neue Matrix Schritt für Schritt. Mit dem + wie unten bei {B}+ wird signalisiert, dass dieser Zustand schon bearbeitet wurde.

Der Startzustand ist a. Von da aus gibt es nur einen a-Übergang nach B.
Von B gibt es keinen a-Übergang, aber b-Übergänge nach A und C.

δ	a	b
{A}	{B}	

δ	a	b
{A}	{B}+	
{B}		{A,C}

Von A und C gibt es a-Übergänge nach A und B.

δ	a	b
{A}	{B}+	
{B}		{A,C}+
{A,C}	{A,B}	

Von A gibt es a-Übergänge nach B, und von B gibt es b-Übergänge nach A und C.

δ	a	b
{A}	{B}+	
{B}		{A,C}+
{A,C}	{A,B}+	
{A,B}	{B}	{A,C}

Jetzt sind alle neuen Zustände bearbeitet. Aus der Matrix kann man dann auch unmittelbar die graphische Darstellung des Automaten rekonstruieren. Dieses Verfahren erzeugt nicht alle Zustände der Potenzmenge der Ausgangszustände. Z.B. wird der Zustand {B,C} nicht erzeugt. Dieser Zustand wäre aber auch nutzlos, da er nicht vom Startzustand aus erreichbar ist.

Theorem 3 (Rabin und Scott) *Der auf diese Weise aus dem nichtdeterministischen endlichen Automaten erzeugte Automat ist deterministisch und akzeptiert die gleiche Sprache wie der Ausgangsautomat.*

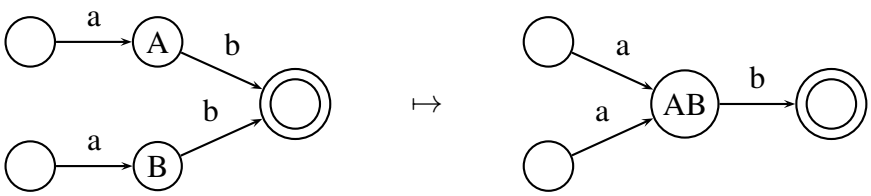
Im Beweis von Rabin und Scott wird ein DEA konstruiert, dessen Zustände aus der gesamten Potenzmenge der Zustände des NEA bestehen. Eingeschränkt auf die vom Startzustand erreichbaren Zustände ergibt sich genau der gleiche Automat wie mit dem obigen Verfahren. Mit dem *Potenzmengenautomaten* ist aber der Beweis einfacher durchführbar. Siehe [1] für Details.

Mit dem NEA als Zwischenschritt können wir nun aus einer regulären Grammatik einen deterministischen endlichen Automaten machen:

Typ 3-Grammatik $\mapsto$ NEA $\mapsto$ DEA.

3.2 Minimalisierung von Automaten

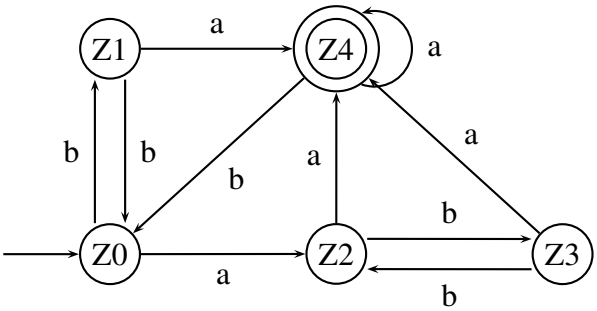
Leider sind die so erzeugten Automaten oft noch zu umständlich. Sie erhalten zu viele redundante Übergänge. Um die Automaten zu verkleinern, untersucht man Zustände, *die auf gleiche Weise* zu einem Endzustand führen. Diese kann man zusammenfassen.



In dem linken Automaten ist es egal, wie man in die Zustände A oder B kommt. Die Zustände A und B sind *äquivalent* in dem Sinn, dass man mit dem gleichen Buchstaben in beide hineinkommt, und von beiden mit dem gleichen Buchstaben b in den Endzustand kommt. Daher kann man die beiden Zustände A und B zu einem Zustand AB zusammenfassen.

Der Algorithmus zur Minimalisierung von DEAs untersucht systematisch alle Zustandspaare und markiert diejenigen, die *nicht* äquivalent sind, d.h. die nicht auf gleiche Weise in einen Endzustand führen. Die am Ende nicht markierten Zustände sind dann äquivalent und werden zusammengefasst.

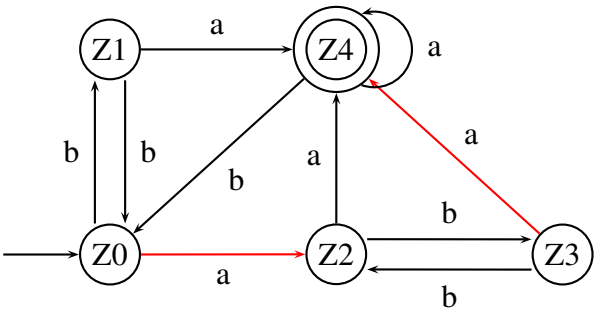
Wir illustrieren das an einem etwas umfangreicheren Beispiel. Für den Automaten auf der linken Seite erzeugt man eine Matrix, mit Spalten und Zeilen für die Zustände. In die Zellen der Matrix markiert man, welche Zustandspaare *nicht* verschmolzen werden können. Die Diagonale bezeichnen die Zustandspaare (Z,Z), die natürlich nicht verschmolzen werden (rote Zellen). Die Matrix ist symmetrisch. Daher braucht man nur eine Dreiecksmatrix auszufüllen. Z4 ist der Endzustand, welcher mit keinem anderen Zustand verschmolzen werden kann. Die entsprechenden Zellen sind dann schon markiert.



Z0					
Z1					
Z2					
Z3					
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

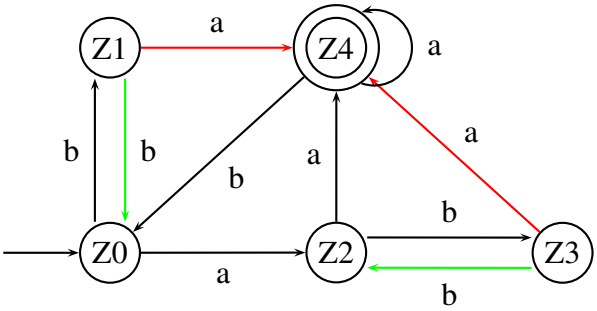
Die weiteren Schritte sind jetzt folgendermaßen: Für jede freie Zelle für das Zustandspaar (X,Y) sucht man einen Buchstaben x mit (X,Y) $\overset{x}{\mapsto}$ (X',Y'). Falls das Paar (X',Y') als nicht äquivalent markiert ist, dann wird auch (X,Y) als nicht äquivalent markiert.

Fangen wir mit (Z3,Z0) $\overset{a}{\mapsto}$ (Z4,Z2). (Z4,Z2) ist markiert, also wird auch (Z3,Z0) markiert.



Z0					
Z1					
Z2					
Z3	X				
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

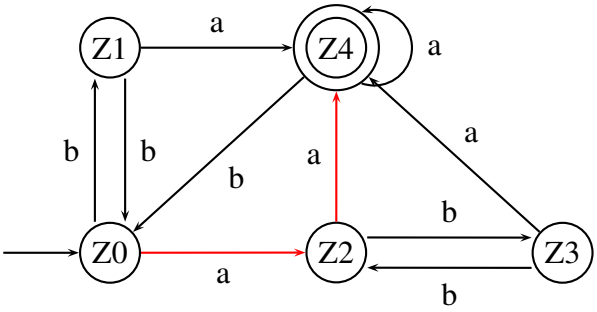
(Z3,Z1) $\overset{a}{\mapsto}$ (Z4,Z4) nicht markiert.
(Z3,Z1) $\overset{b}{\mapsto}$ (Z2,Z0) nicht markiert.
Daher können wir keine Markierung setzen.



Z0					
Z1					
Z2					
Z3	X				
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

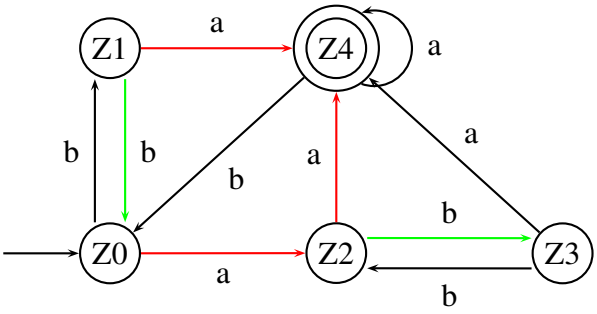
(Z3,Z2) $\overset{a}{\mapsto}$ (Z4,Z4) ist nicht markiert.
(Z3,Z2) $\overset{b}{\mapsto}$ (Z2,Z3) ist nicht markiert.

(Z2,Z0) $\overset{a}{\to}$ (Z4,Z2) ist markiert.



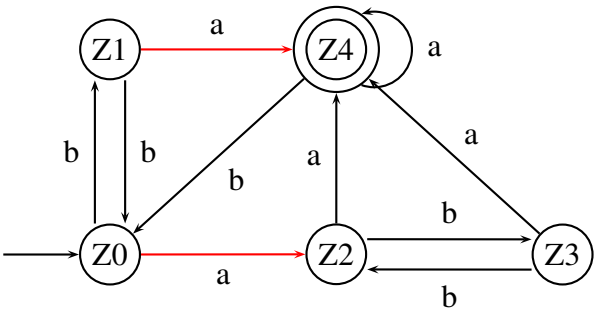
Z0					
Z1					
Z2	X				
Z3	X				
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

(Z2,Z1) $\overset{a}{\mapsto}$ (Z4,Z4) ist nicht markiert.
(Z2,Z1) $\overset{b}{\mapsto}$ (Z3,Z0) ist markiert.



Z0					
Z1					
Z2	X	X			
Z3	X				
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

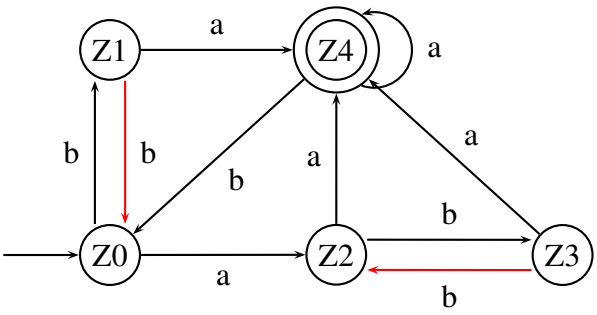
(Z1,Z0) $\overset{a}{\mapsto}$ (Z4,Z2) ist markiert.



Z0					
Z1	X				
Z2	X	X			
Z3	X				
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

Es gibt noch die zwei unmarkierten Zellen. Da die Markierung von den anderen Markierungen abhängt, muss man diese ein zweites mal überprüfen. Das muss man so lange wiederholen, bis sich keine Änderung mehr ergibt (Fixpunktiteration).

(Z3,Z1) $\overset{a}{\mapsto}$ (Z4,Z4) ist nicht markiert.
(Z3,Z1) $\overset{b}{\mapsto}$ (Z2,Z0) ist markiert.

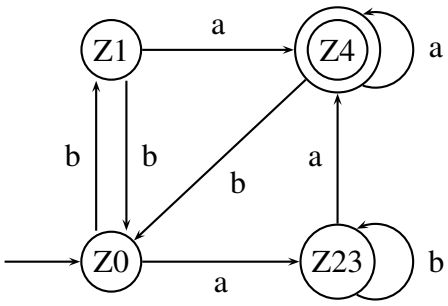


Z0					
Z1	X				
Z2	X	X			
Z3	X	X			
Z4	X	X	X	X	
	Z0	Z1	Z2	Z3	Z4

(Z3,Z2) $\overset{a}{\mapsto}$ (Z4,Z4) ist nicht markiert.
(Z3,Z2) $\overset{b}{\mapsto}$ (Z2,Z3) ist nicht markiert.

D.h. das einzige Paar, welches offen bleibt ist (Z3,Z2). Diese Zustände sind äquivalent und können verschmolzen werden.

Der resultierende Minimalautomat ist nun



Theorem 4 (Myhill, Nerode) *Das skizzierte Verfahren berechnet den Minimalautomaten. Der Minimalautomat ist eindeutig bis auf Isomorphie.*

Der Beweis führt zunächst eine Äquivalenzrelation auf Zuständen ein und zeigt dann, dass die Zustände des Minimalautomaten die Äquivalenzklassen dieser Äquivalenzrelation sind. Für Details siehe [1]

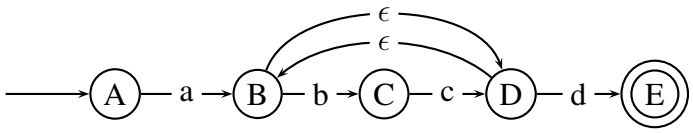
Eine optimierte Vorgehensweise für das Parsen von Typ 3-Sprachen ist nun:

$$\text{Typ 3-Grammatik} \quad \mapsto \quad \text{NEA} \quad \mapsto \quad \text{DEA} \quad \mapsto \quad \text{Minimalautomat}$$

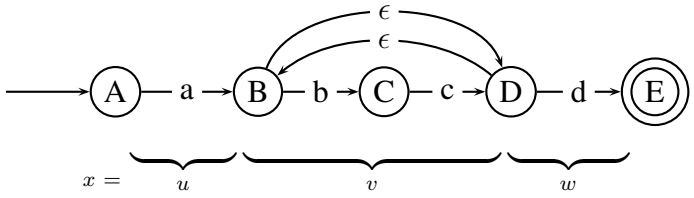
4 Das Pumping Lemma

Wir haben gesehen, dass Typ 3-Sprache durch endliche deterministische Automaten geparst werden können. Die meisten Typ 3-Sprachen haben allerdings unendlich viele, und zwar endlich, aber beliebig lange Wörter, wogegen die Automaten nur endlich viele Zustände haben. Betrachten wir die Sprache $L = \{a(bc)^i d \mid i \geq 0\}$. Die Wörter dieser Sprache sind $\{ad, abcd, abcbcd, abcbcbcd, \ldots\}$ Für jeden Automaten gibt es dann Wörter, die länger sind als der Automat Zustände hat. So ein Automat kann diese langen Wörter nur akzeptieren wenn er einen Zyklus hat.

Im folgenden Automaten ist es der Zyklus B $\mapsto$ C $\mapsto$ D $\mapsto$ B:



Jedes akzeptierte Wort hat also einen Durchlauf durch den Automaten, der aus drei Phase besteht: Anfangsteil bis zum Zyklusanfang, Durchlauf durch den Zyklus, Endteil aus dem Zyklus heraus. Der Durchlauf durch den Zyklus kann auch übersprungen werden. Man kann also ein Wort x, welches mehr Buchstaben hat als der Automat Zustände, aufteilen in drei Teile $x = uvw$ wobei u der Anfangsteil in den Zyklus darstellt, v den Zyklusteil, der beliebig oft wiederholt werden kann, und w der Endteil, aus dem Zyklus heraus. u und w können auch leer sein, aber der Zyklusteil muss mindestens einen Buchstaben enthalten.



Dies gilt für alle Typ 3-Sprachen. Es ist der Inhalt des *Pumping Lemmas*:

Lemma 1 (Pumping Lemma) *Für jede Typ 3-Sprache L gilt: Es gibt eine* Pumping Zahl *n, so dass jedes Wort x mit $|x| > n$ zerlegt werden kann in drei Teile $x = uvw$ mit den Eigenschaften:*

1. $|v| > 0$
2. $|uv| < n$
3. $uv^iw \in L$ *für alle* $i \geq 0$.

Die Pumping Zahl entspricht der Anzahl Zustände im Minimalautomaten für L, kann aber im Einzelfall auch noch kleiner sein. Bedingung 1 sagt, dass der Zyklusteil nicht leer sein darf. Bedingung 2 sagt, dass der Anfangsteil bis zum Ende des Zyklus kleiner als die Pumpingzahl sein muss. Schließlich ist Bedingung 3 genau der Pumpeffekt: man kann den Mittelteil beliebig oft (incl. 0 mal) „aufpumpen".

Das Pumping Lemma direkt ist nicht von allzu großem Nutzen. Es gibt nämlich auch Sprachen von höherem Typ, für die das Pumping Lemma auch gilt. $L = \{a^nb^mc^m \mid n, m > 0\}$ ist eine Typ 2-Sprache, für die das Pumping Lemma auch gilt.

Wir können aber die Aussage:

Sprache ist vom Typ 3 $\Rightarrow$ Pumping Lemma gilt.

in seine Kontraposition verkehren:

Pumping Lemma gilt *nicht* $\Rightarrow$ Sprache ist *nicht* vom Typ 3.

Ein Beweis, dass das Pumping Lemma nicht gilt, macht man am besten durch Widerspruch:

(Annahme: Pumping Lemma gilt $\Rightarrow$ Widerspruch)
$\Rightarrow$ Pumping Lemma gilt nicht
$\Rightarrow$ Sprache ist nicht vom Typ 3.

Wie leitet man nun einen Widerspruch aus der Annahme, dass das Pumping Lemma für die Sprache L gilt her? Die Methode ist folgendermaßen:

- Man postuliert eine Pumping Zahl n (die existieren muss, die wir aber nicht kennen).
- Man wählt ein Wort x mit $|x| > n$.
  Hier muss man kreativ sein, um ein geeignetes Wort zu finden.
- Man untersucht *alle möglichen* Zerlegungen $x = uvw$,
  und zeigt für *jede davon*, dass für mindestens ein i: $uv^iw \notin L$ ist.

Wir demonstrieren das am Beispiel der Sprache $L = \{a^mb^m \mid m > 0\}$.

Die Formulierung des Beweises wäre nun:

- Sei n die Pumping Zahl.
- Wir wählen das Wort $x = a^n b^n$.
- Aus der Bedingung $|uv| < n$ folgt, dass uv nur aus a's bestehen kann.
  (Wenn z.B. $n = 5$ wäre, dann hätten wir: $x = \underbrace{aaaaa}_{uv}bbbbb$, und zwar egal, wie die Zerlegung ist)
- Damit werden nur Buchstaben a „aufgepumpt", aber nicht b. Für jede mögliche Zerlegung erhalten wir daher z.B. $uv^2w \notin L$ (da mehr a's als b's)
- Das ist eine Widerspruch zur Aussage des Pumping Lemmas.

Also ist L nicht vom Typ 3.

Um Wörter aus L zu parsen, müsste man die a's zählen, und dann mit der Anzahl b's vergleichen. Ein endlicher Automat hat aber keinen Zählmechanismus. Daher kann L nicht vom Typ 3 sein.

Heuristik: Generell kann man sagen, alle Sprachen, bei denen man sich beim Parsen etwas merken muss, können nicht vom Typ 3 sein. Typische Beispiele sind Sprachen mit öffnenden und schließenden Klammern.

Wir demonstrieren es am Beispiel einfacher arithmetischer Ausdrücke:

Die Grammatik für L sei:

$A \to 0 \mid \ldots \mid 9 \mid y \mid z \mid (A + A) \mid (A * A)$

Wörter in dieser Sprache sind z.B. $((y * 3) + (z * 5)))$. Hier brauchen wir gleich viele öffnende wie schließende Klammern.

Die Grammatik selbst ist offensichtlich nicht vom Typ 3, sondern vom Typ 2 . Das schließt aber prinzipiell nicht aus, dass es auch eine Grammatik vom Typ 3 geben könnte. Um das auszuschließen, brauchen wir das Pumping Lemma.

Nach dem Beweisschema von oben formulieren wir also folgendermaßen

- Sei n die Pumping Zahl.
- Wir wählen das Wort $x = \underbrace{(\ldots(}_{n} y + y) + \ldots + y)$.
- Aus der Bedingung $|uv| < n$ folgt, dass uv nur aus (bestehen kann.
- Damit werden nur öffnende Klammern „aufgepumpt", aber nicht die schließenden Klammern. Für jede mögliche Zerlegung erhalten wir daher z.B. $uv^2w \notin L$ (da mehr (als))

- Das ist eine Widerspruch zur Aussage des Pumping Lemmas.

Also ist L nicht vom Typ 3.

Die Beweise mit dem Pumping Lemma stehen in einer ganzen Reihe von *Unmöglichkeitsbeweisen* in der theoretischen Informatik:
Es ist unmöglich, für die Sprache eine Typ 3-Grammatik zu finden.

5 Weitere Entscheidbarkeitsprobleme

Wir haben gesehen, dass das Wortproblem für Typ 3-Sprachen entscheidbar ist, sogar in *linearer Zeit* ($\mathcal{O}(n)$), da man nur einmal durch das Wort iterieren muss.

Weitere Probleme sind:

Das Leerheitsproblem ist entscheidbar. Man testet ob es in dem Automaten überhaupt einen Pfad vom Anfangszustand zu einem Endzustand gibt.

Das Endlichkeitsproblem ist die Frage, ob die Sprache nur endlich viele Wörter hat. Das ist genau dann der Fall wenn der Minimalautomat keinen Zyklus hat. Das kann man in linearer Zeit feststellen.

Das Äquivalenzproblem ist die Frage, ob zwei Typ 3-Sprachen gleich sind. Da die Minimalautomaten eindeutig sind, lässt sich das entscheiden, indem man die beiden Minimalautomaten auf Isomorphie testet (dabei spielt die Benennung der Zustände keine Rolle).

Neben den Typ 3-Grammatiken gibt es einen weiteren Spezifikationsmechanismus für Typ 3-Sprachen: *reguläre Ausdrücke.* Da diese von großem praktischen Nutzen sind, werden in einem separaten Text besprochen.

Stichwortverzeichnis

Literatur

[1] Uwe Schöning. *Theoretische Informatik – kurzgefasst*. Hochschultaschenbuch. Akademischer Verlag, 5. auflage edition, 2008. 5. Auflage.