IN DIESEM KAPITEL

- lernen Sie sogenannte Deterministische Endliche Automaten, kurz DFAs, kennen
- erfahren Sie wie ein DFA funktioniert
- lernen Sie die Möglichkeiten und Grenzen von DFAs kennen

Kapitel 1
Deterministische Endliche Automaten (DFAs)

DFAs stellen die einfachsten mathematischen Modelle für unsere Computer dar. Diese Modelle bekommen als Input einen String aus Eingabezeichen. Der Output besteht aus einer Entscheidung darüber, ob das Wort »akzeptiert« wird oder nicht, d. h., der Automat kann erkennen, ob das Eingabewort eine bestimmte Eigenschaft besitzt oder nicht. In diesem Kapitel werden wir die Möglichkeiten und Grenzen dieser Automaten erkunden.

Einführung

»The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform.«

(Die Analytical Engine hat keinerlei Ambitionen, irgendetwas entstehen zu lassen. Sie kann die Dinge tun, von denen wir wissen, wie wir ihr befehlen können, sie auszuführen.) »Ada Lovelace, in »Notes of the Translator«, Taylor's Scientific Memoirs, 1843. Dieses Zitat zum näheren Einstieg. Das Verhalten unserer Computer wird von zwei Faktoren bestimmt: Ihrem inneren *Zustand*, d. h. den Bits und Bytes in ihren Speicherzellen, die die aktuelle Konfiguration des Computers bestimmen, und den *Eingaben*, die wir als User machen. Im Folgenden versuchen wir, einfache mathematische Modelle für Computer zu finden und so ihr Verhalten besser zu verstehen. Dabei müssen wir darauf achten, dass wir uns von möglichst vielen unnötigen Details befreien (z. B. sollte es für unsere Zwecke egal sein, ob die Eingaben von einer Tastatur oder von einer Maus kommen). Anderenfalls wird unser Modell zu kompliziert, und es wird zu schwierig, allgemeine Aussagen über ihr Verhalten zu machen. Ist unser Modell auf der anderen Seite zu einfach, so sind die Möglichkeiten des Modells zu stark eingeschränkt, und wir können nicht erwarten, über das Modell interessante Informationen zu erhalten, da es zu stark von der Wirklichkeit abweicht.

Das einfachste dieser Modelle bilden die **Deterministischen Endlichen Automaten** (engl. Deterministic Finite Automata, DFAs). In ihnen setzen wir die Eigenschaft realer Computer um, bei Eingaben von außen ihren Zustand zu wechseln. Das Wort *Determinismus* kommt aus dem Lateinischen und bedeutet so viel wie »vorherbestimmt«. DFAs verarbeiten eine Folge von Eingabezeichen und setzen diese anhand eines vorgegebenen Regelwerks (man könnte auch sagen, Programms) in Wechsel ihres inneren Zustands um. Deterministische Automaten besitzen die Eigenschaft, dass sich ihr zukünftiges Verhalten anhand ihres aktuellen Zustands und der Eingabezeichen vorhersagen lässt.

Niemand würde auf die Idee kommen, abstrakte DFAs, wie wir sie hier diskutieren, tatsächlich zu bauen. Dennoch hilft es vielleicht Ihrer Intuition, einen Hardware-DFA bildlich vor sich zu sehen:

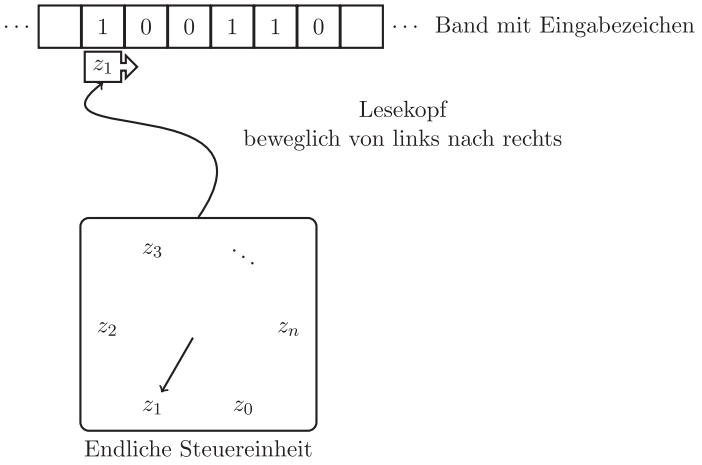


Abbildung 1.1: Hardware-Modell eines Deterministischen Endlichen Automaten

Der Automat liest die Folge von Eingabezeichen von einem Eingabeband. Der Lesekopf kann sich immer nur um ein Feld weiter und nur in einer Richtung bewegen. Eine Steuereinheit verarbeitet die Eingabezeichen und veranlasst die Zustandswechsel. Wichtig dabei: Die Anzahl der Zustände der Steuereinheit ist **endlich** (so wie auch unsere Computer nur endlich viele Speicherzustände besitzen) und die Zustandswechsel sind, wie schon gesagt, bei Kenntnis des vorherigen Zustands und des Eingabezeichens vorhersagbar, also **deterministisch**.

Erste Beispiele

DFAs lassen sich zur Modellierung vieler alltäglicher Situationen benutzen, die auf den ersten Blick nichts mit Computern zu tun haben. Wir beginnen hier mit einigen Beispielen, bevor wir im nächsten Abschnitt eine formale Definition eines DFAs geben.

Ein Aufzug kann als DFA betrachtet werden. Die Zustände des Automaten sind dabei die Stockwerke, die der Aufzug anfahren kann. Die Eingabe in den Automaten stellt das gewünschte Stockwerk dar. Sie erfolgt über Tastendrücke im Aufzug oder über die »Holtaste«. Je nachdem, ob der Aufzug über eine Stockwerksanzeige verfügt oder nicht, handelt es sich um einen DFA mit bzw. ohne Ausgabe.

Unser nächstes Beispiel stellt ein altes Rätsel für Kinder dar. Vielleicht haben Sie schon einmal davon gehört:

Ein Bauer B, ein Wolf W, eine Ziege Z und ein Kohlkopf K stehen am linken Ufer eines Flusses. Es gibt zwar ein Boot, in das aber nur der Bauer und ein weiteres Objekt hineinpassen. Zudem sollten der Wolf und die Ziege bzw. Ziege und Kohlkopf niemals allein an einem Ufer stehen, da es sonst zu unerwünschten Nebeneffekten kommt. Wie können alle vier sicher den Fluss überqueren? Auch dieses alte Problem lässt sich mit Hilfe eines DFAs (ohne Ausgabe) modellieren. Die Zustände sind dabei die Menge der Objekte, die sich aktuell am linken Ufer befinden. Da zu Beginn noch alle am linken Ufer sind, ist das System am Anfang also im sogenannten **Startzustand** $z_0 = \{B, W, Z, K\}$.

Die Eingaben in das System bestehen aus den Objekten, die der Bauer mit zu sich ins Boot holt, kommen also aus der Menge $\Sigma = \{w, z, k, \square\}$. Der Bauer muß immer mitfahren (sonst ließe sich das Boot nicht steuern) und wird deshalb nicht eigens aufgeführt. Das Zeichen $\square$ steht für den Fall, dass der Bauer alleine fährt. Der gewünschte *Endzustand* $z_e = \{\}$ ist erreicht, wenn alle Objekte am anderen Ufer sind.

Nach diesen Vorbereitungen ist es nicht allzu schwer, sich zu überlegen, wie der Bauer vorgehen muss. Er muss bei den Überfahrten lediglich darauf achten, dass das System nicht in einen der verbotenen Zustände $\{W, Z\}, \{Z, K\}, \{W, Z, K\}, \{B, K\}, \{B, W\}, \{B\}$ gerät. Visualisiert man sich die aus den **erlaubten** Bootsfahrten resultierende Abfolge der Zustände und lässt verbotene Zustände und Überfahrten, die in verbotene Zustände führen, der Übersichtlichkeit halber weg, erhält man den gerichteten Graphen in Abbildung 1.2. Der Endzustand wurde doppelt umrandet.

Die Frage ist nun: Wie kommen wir vom Start- in den Endzustand? Aus Abbildung 1.2 lässt sich ablesen, dass eine Abfolge von Zügen der Form $z\square wzk\square z$ oder $z\square kzw\square z$ den Bauern zum erwünschten Ziel führt. Dies sind aber längst nicht die einzig möglichen Lösungen: Hat der Bauer Spaß am Boot fahren, kann er auch nach dem Schema $z\square(wzk)(wzk)\ldots(wzk)\square z$ bzw. $z\square(kzw)(kzw)\ldots(kzw)\square z$ vorgehen, solange die Anzahl der Wiederholungen von wzk bzw. kzw ungerade ist. Wir werden später sehen, dass die Möglichkeit, Teile derjenigen Eingabewörter, die den Automaten in den Endzustand überführen, wiederholen zu können, typisch für DFAs ist.

Das letzte Beispiel enthält bereits alle wichtigen Zutaten der formalen Definition eines DFAs: Es gibt einen Start- und einen Endzustand, und wir sehen, dass sich die

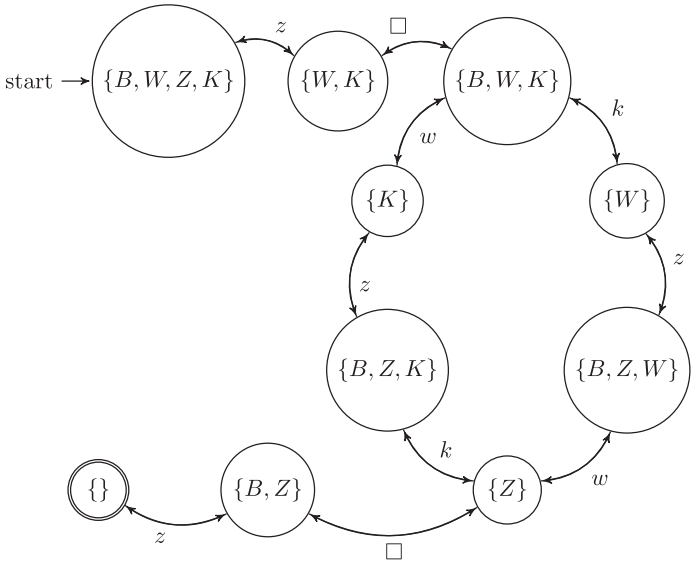


Abbildung 1.2: Die Geschichte vom Bauern, dem Wolf, der Ziege und dem Kohlkopf

Zustandsübergänge eines DFA mit Hilfe eines Graphen gut visualisieren lassen. Dieser Graph bildet gewissermaßen die Software bzw. das Räderwerk im Innern der Steuereinheit des DFA s.

Unser letztes einführendes Beispiel kommt aus der IT-Welt: Netzwerkprotokolle lassen sich in der Regel gut über DFAs veranschaulichen, da sie ihre Zustände nach genau vorgeschiebenen (d. h. standardisierten) Regeln wechseln. Wäre das anders, wäre die Kommunikation zwischen Rechnern, die zuvor noch keinen Kontakt hatten, sehr mühsam.

Ein Server, der im Internet einen Dienst anbietet, erzeugt dazu zunächst einen so genannten *Socket* (Sockel) und wartet dann auf Verbindungswünsche von Clients über das TCP-Protokoll. Der Socket besteht aus der IP-Adresse des Servers und der Portnummer des angebotenen Diensts (z.B. 80 für http). Auch Clients können Sockets mit selbst gewählten Portnummern erzeugen und von dort aus Verbindungen zu Servern initiieren. Dabei läuft zunächst der sogenannte *Three-Way-Handshake* des TCP-Protokolls ab, bei dem ein Verbindungsaufbauwunsch (zu erkennen am gesetzten SYN-Flag) mit einem SYN-ACK-Paket (SYN- und ACK-Flag gesetzt) beantwortet wird. Ein finales ACK-Paket beendet den Handshake.

Nach geglücktem Verbindungsaufbau werden TCP-Pakete mit gesetztem ACK-Flag zwischen Client und Server ausgetauscht, bis die Verbindung von einer der Parteien mit einem FIN-Paket beendet wird. Jedes Mal wenn ein Paket gesendet oder empfangen wird, wechselt der Socket in genau vorherbestimmter Weise seinen Zustand. Sein Verhalten kann also als DFA modelliert werden, wobei die Zustandsübergänge durch die Flags (das sind bestimmte wohldefinierte Bits im Header der Pakete, die die Art des Pakets festlegen) der empfangenen Pakete ausgelöst werden.

Grundlegende Definitionen

Wir sind nun bereit, einen DFA als mathematisches Modell formulieren zu können. Dazu müssen wir die Zustände, die Eingaben und die Reaktion des DFA auf die Eingaben modellieren.

Symbole und Wörter

Wir beginnen mit der Modellierung der Eingaben in den DFA, also den Zeichen bzw. Zeichenketten auf dem Eingabeband in Abb. 1.1:

$\Sigma \neq \emptyset$ sei eine endliche Menge, das **Alphabet**. Die Elemente von Σ heißen **Symbole** oder **Zeichen**.

Ein **Wort** oder **String** w ist eine endlich lange Folge von Symbolen. Die Menge aller Wörter, die aus den Symbolen in Σ gebildet werden können, wird mit Σ^* bezeichnet.

Die **Länge** eines Worts w wird mit $|w|$ bezeichnet. $|w|$ steht für die Anzahl der Symbole, aus denen w besteht. Σ^* enthält auch ein besonderes Wort, das so genannte **leere Wort** ϵ. Es besteht aus keinem Symbol, hat also die Länge null.

Eine Teilmenge $L \subseteq \Sigma^*$ von Wörtern heißt **formale Sprache** über Σ.

Sie kennen natürlich den Begriff des Alphabets aus dem Alltag. Die englische Sprache kommt mit dem Alphabet $\Sigma = \{a, b, c \ldots, z\}$ aus. Die Menge Σ^* enthält sinnvolle Wörter wie *year* oder *house*, aber natürlich auch sprachlich sinnlose Strings wie *qugh* oder *cxab*. Da wir in diesem Buch nicht an menschlichen Sprachen interessiert sind, unterscheiden wir an dieser Stelle nicht zwischen sinnvollen und sinnlosen Strings.

Typische Alphabete, die wir in diesem Buch immer wieder benutzen werden, sind $\Sigma = \{0, 1\}$ und $\Sigma = \{a, b\}$. Für $\Sigma = \{0, 1\}$ ist $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \ldots\}$. Offenbar lässt sich aus einer endlichen Menge von Symbolen eine abzählbar unendliche Menge[1] von Wörtern formen. Das gilt sogar, wenn Σ nur aus einem Symbol besteht: Für $\Sigma = \{a\}$ zum Beispiel ist $\Sigma^* = \{\epsilon, a, a^2, a^3, a^4, \ldots\}$.

Oft werden wir zwei Wörter u und v, die aus derselben Menge Σ^* kommen, einfach aneinander hängen: Aus $u = a_1 a_2 \ldots a_n$ und $v = b_1 b_2 \ldots b_m$ wird das Wort $w = uv = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m$. Diese Operation bezeichnen wir als **Konkatenation**. Natürlich gilt der Zusammenhang $|u| = n, |v| = m \Rightarrow |uv| = n + m$. Voranstellen oder Anhängen des leeren Worts ϵ an ein Wort u ändert nichts: $\epsilon u = u\epsilon = u$. Konkateniert man ein Wort u mit sich selbst, erhält man das Wort $u^2 = uu$. Allgemeiner setzen wir $u^n = \underbrace{uu \cdots u}_{n \text{ mal}}$. Ist also z. B. $\Sigma = \{a, b\}$ und $u = abb$, so ist $u^4 = (abb)^4 = abbabbabbabb$.

[1] Die Elemente abzählbar unendlicher Mengen lassen sich nummerieren, d. h., eine abzählbar unendliche Menge ist so groß wie die Menge $\mathbb{N}$ der natürlichen Zahlen.

Die Definition eines DFAs

Wir können nun die komplette Definition eines DFAs angeben. Zentrales Element wird dabei die so genannte **Übergangsfunktion** sein, die die Zustandswechsel des Automaten als Reaktion auf ein Eingabezeichen beschreibt. Lassen Sie sich von dem mathematischen Formalismus nicht abschrecken. Er ist nötig, weil es sich bei einem DFA um ein mathematisches Modell handelt. Nur durch einen gewissen Grad an Abstraktion ist es möglich, sich von überflüssigen oder vom wesentlichen ablenkenden Details (wie z.B. der Frage, wie ein DFA in Hardware realisiert ist) zu befreien. Mit zunehmender Beschäftigung mit DFAs und ihren Varianten werden Sie auch an Vertrautheit mit dem Formalismus gewinnen.

Ein **Deterministischer Endlicher Automat (DFA)** ist ein 5-Tupel $M = (Z, \Sigma, \delta, z_0, F)$, bestehend aus:

- ✔ einer endlichen **Zustandsmenge** Z, die die Zustände enthält.
- ✔ einem **Eingabealphabet** Σ, wobei $Z \cap \Sigma = \emptyset$.
- ✔ einer **Übergangsfunktion** $\delta : Z \times \Sigma \to Z$.
- ✔ einem **Startzustand** $z_0 \in Z$.
- ✔ einer Menge von **Endzuständen** $F \subseteq Z$.

Lassen Sie uns die fünf Zutaten eines DFAs nacheinander durchsprechen. Die endliche **Zustandsmenge** Z ist für das D (wie *finite*) in DFA verantwortlich. Auch die Automaten, denen wir im Alltag begegnen, kennen nur eine endliche Anzahl von Zuständen.

Das **Eingabealphabet** besteht aus den Zeichen, die in den Automaten eingegeben werden können und damit Zustandsübergänge bewirken. Natürlich müssen wir deutlich zwischen Eingabezeichen auf der einen Seite und Zuständen auf der anderen Seite unterscheiden, deshalb die Forderung $Z \cap \Sigma = \emptyset$.

Die **Übergangsfunktion** δ beschreibt das Verhalten des Automaten, d. h., die Zustandsübergänge, wenn Symbole in den Automaten eingegeben werden. Der neue Zustand hängt sowohl vom alten Zustand als auch vom Eingabesymbol ab. Deshalb kommt der Input für die Funktion δ aus der Menge $Z \times \Sigma$, also der Menge der geordneten Paare von Zuständen und Eingabesymbolen. Der Output hingegen ist ein einzelner, genau festgelegter Zustand z aus der Zustandsmenge Z. Ein allgemeiner Zustandsübergang hat also die Form

$$\delta(z_{alt}, a) = z_{neu}$$

für ein Symbol $a \in \Sigma$ und $z_{neu}, z_{alt} \in Z$. Beispielsweise lautet einer der Übergänge des Bauer-Ziege-Wolf-Kohlkopf-Automaten oben:

$$\delta(\{B, W, K\}, w) = \{K\}.$$

Eine sogenannte **Wertetabelle** listet alle diese Zustandsübergänge auf. Im Prinzip lässt sich die Übergangsfunktion und damit das Verhalten des Automaten durch eine Wertetabelle komplett beschreiben. So eine Liste wird aber schnell unübersichtlich, und das Verhalten

des Automaten lässt sich nur schwer daraus ablesen. Zum Glück lässt sich ein DFA mit Hilfe eines gerichteten Graphen auch grafisch darstellen, wie wir am Beispiel des Bauer-Ziege-Wolf-Kohlkopf-Automaten gesehen haben. Dabei entsprechen die Knoten des Graphen den Zuständen und die Pfeile den Zustandsübergängen. Der Graph wird auch als **Übergangsdiagramm** des DFAs bezeichnet.

Schließlich besitzt jeder DFA einen speziellen **Startzustand** z_0, in dem er sich befindet, bevor irgendwelche Eingaben gemacht werden, und einen oder mehrere **Endzustände**, die gewissermaßen das Ziel des Automaten vorgeben. Wie das folgende Beispiel zeigt, können Start- und Endzustand durchaus zusammenfallen.

Wir betrachten den DFA

$M = (Z, \Sigma, \delta, z_0, F)$ mit $Z = \{z_0, z_1\}, \Sigma = \{0, 1\}, F = \{z_0\}$.

Die Übergangsfunktion δ des Automaten wird durch die folgende Wertetabelle definiert:

δ	0	1
z_0	z_1	z_0
z_1	z_0	z_1

Der dazugehörige Übergangsgraph sieht damit so aus wie in Abbildung 1.3.

Was macht dieser Automat? Anders gefragt, welche Strings überführen den Startzustand in den Endzustand? Da der Startzustand gleichzeitig auch Endzustand ist, brauchen wir gar nichts einzugeben und sind schon gewissermaßen am Ziel. Das leere Wort ϵ gehört also sicherlich zu den gesuchten Strings. Ist das Eingabewort nicht leer, so bewirkt jede 0 einen Sprung über den (gedachten) Graben zwischen z_0 und z_1. Zwei Sprünge (oder ein Vielfaches davon) bringen uns wieder zurück nach z_0. Also bewirken alle Strings mit der Eigenschaft, eine gerade Anzahl von 0 zu enthalten, dass der Automat vom Start- in den Endzustand übergeht. Das leere Wort ϵ passt auch in dieses Schema, da es null mal das Symbol 0 enthält. Man kann also sagen: Der Automat in Abbildung 1.3 prüft, ob ein Eingabewort eine gerade Anzahl von 0 enthält.

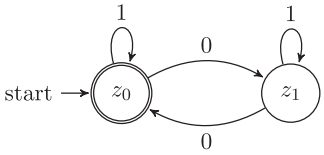


Abbildung 1.3: Ein erstes Übergangsdiagramm eines DFA

Reguläre Sprachen

Die Frage, welche Menge von Wörtern (bzw. welche **Sprache**) einen DFA vom Start- in den Endzustand überführt, ist eine typische Aufgabenstellung in der Automatentheorie. In diesem Abschnitt werden wir weitere Beispiele für solche Mengen anschauen und untersuchen, welche Eigenschaften sie haben.

Die erweiterte Übergangsfunktion

Die Übergangsfunktion δ sagt uns, wie ein Automat auf einzelne Eingabezeichen reagiert. Wenn wir aber wissen möchten, welche *Wörter* einen Automaten vom Start- zum Endzustand überführen, wäre eine Art erweiterte Übergangsfunktion nützlich, die uns die Reaktion des Automaten auf ganze Eingabewörter liefert. Welche Eigenschaften müsste eine solche erweiterte Übergangsfunktion $\hat{\delta}$ (sprich: Delta-Dach) haben?

Zunächst einmal bekommt $\hat{\delta}$ ein Paar aus einem Zustand und einem Wort als Input, das heißt, $\hat{\delta}$ muss auf der Menge $Z \times \Sigma^*$ operieren. Der Output $\hat{\delta}(z, w)$ ist dann derjenige Zustand, in den der DFA vom Zustand z nach Eingabe des Worts w übergeht. Insgesamt muss also gelten:

$$\hat{\delta} : Z \times \Sigma^* \to Z$$

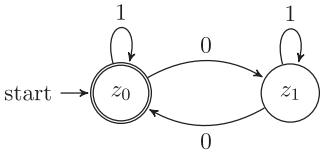
Gibt man das leere Wort in den Automaten ein, soll sich am aktuellen Zustand nichts ändern:

$$\forall z \in Z : \hat{\delta}(z, \epsilon) = z$$

Außerdem darf es natürlich keine Konflikte mit der bestehenden Übergangsfunktion δ geben. Gibt man nur ein einziges Symbol ein, so muss der Output von $\hat{\delta}$ mit dem von δ übereinstimmen. Diese Überlegung lässt sich dazu ausnutzen, die Wirkung von $\hat{\delta}$ mit Hilfe von δ rekursiv zu definieren:

$$\forall z \in Z, w \in \Sigma^*, a \in \Sigma : \hat{\delta}(z, wa) = \delta(\hat{\delta}(z, w), a)$$

Tatsächlich reichen diese beiden Eigenschaften aus, um $\hat{\delta}$ vollständig festzulegen. Als Beispiel betrachten wir noch einmal den Automaten aus Abb. 1.3, also



und berechnen $\hat{\delta}(z_0, 010)$:

$$\begin{aligned}\hat{\delta}(z_0, 010) &= \delta(\hat{\delta}(z_0, 01), 0) = \delta(\delta(\hat{\delta}(z_0, 0), 1), 0) \\ &= \delta(\delta(\delta(\hat{\delta}(z_0, \epsilon), 0), 1), 0) \\ &= \delta(\delta(\delta(z_0, 0), 1), 0) = \delta(\delta(z_1, 1), 0) = \delta(z_1, 0) = z_0\end{aligned}$$

Das erscheint unnötig aufwendig und kompliziert. Wir haben zwar das erwartete Ergebnis z_0 bekommen, aber warum kann man nicht einfach bei z_0 starten und das Eingabewort dann Symbol für Symbol von links nach rechts mit Hilfe von δ abarbeiten? Tatsächlich wird man, wenn man bei einem konkreten DFA $\hat{\delta}(z, w)$ für einen konkreten Zustand z und ein konkretes Wort w ausrechnen möchte, immer das Wort w von links nach rechts bearbeiten, ohne auf die Definition von $\hat{\delta}$ zurückzugreifen.

Aber für abstrakte Argumentationen in Beweisen oder Defintionen, die nicht einem konkreten Wort arbeiten, ist es Gold wert, die erweiterte Übergangsfunktion $\hat{\delta}$ als formales Werkzeug zur Hand zu haben. Dies zeigt schon die nächste Definition, in der wir die Menge der Wörter, die einen DFA vom Start- in den Endzustand überführen, mit Hilfe von $\hat{\delta}$ auf elegante Weise charakterisieren können:

Ein DFA $M = (Z, \Sigma, \delta, z_0, F)$ **akzeptiert** ein Wort $w \in \Sigma^*$, wenn der Anfangszustand z_0 durch w in einen Endzustand überführt wird, d. h., wenn gilt:

$\hat{\delta}(z_0, w) = z \in F.$

Die Menge aller von M akzeptierten Wörter heißt die von M **akzeptierte Sprache** $L(M)$:

$L(M) = \{w \in \Sigma^* | \hat{\delta}(z_0, w) \in F\}$

Eine formale Sprache L heißt **regulär**, falls es einen DFA M gibt, der L als akzeptierte Sprache hat.

Beispiele regulärer Sprachen

Eine reguläre Sprache haben wir bereits kennengelernt: Die Sprache der Wörter mit einer geraden Anzahl von Nullen. Es folgen nun einige weitere Beispiele für reguläre Sprachen mit den dazugehörigen DFAs.

Wir betrachten die Sprache

$L = \{w \in \Sigma^* | w = u00 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$

über dem Alphabet $\Sigma = \{0, 1\}$. Es geht also um die Menge der binären Strings, die auf 00 enden. Ein dazugehöriger Automat M, der L akzeptiert, sieht so aus:

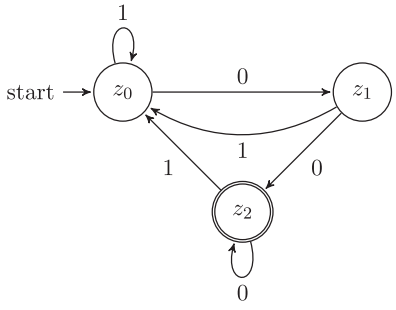


Abbildung 1.4: Ein DFA, der prüft, ob das Eingabewort auf zwei Nullen endet.

Zum Beispiel ist $\hat{\delta}(z_0, 00) = z_2 \in F$ und $\hat{\delta}(z_0, 10) = z_1 \notin F$. Aber woher wissen wir, dass M *genau* die Wörter aus L akzeptiert? Bei diesem einfachen Automaten können wir direkt nachrechnen, dass alle Wörter aus L akzeptiert werden. Dazu prüfen wir mit Hilfe von $\hat{\delta}$ nach, dass alle Wörter der Form $u00$ von z_0 in den Endzustand z_2 führen:

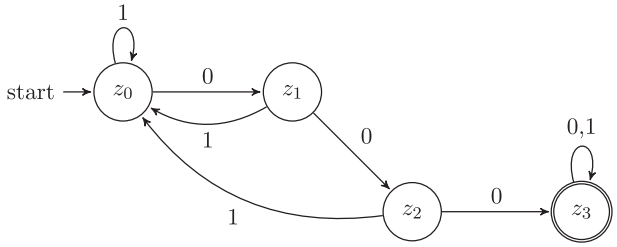
$$\begin{aligned}\hat{\delta}(z_0, u00) &= \delta(\hat{\delta}(z_0, u0), 0)\\ &= \delta(\delta(\hat{\delta}(z_0, u), 0), 0)\\ &= \delta(\delta(z_i, 0), 0), \text{ wobei } 0 \le i \le 2\\ &= \delta(z_j, 0), \text{ wobei } 1 \le j \le 2\\ &= z_2 \in F\end{aligned}$$

Damit ist klar, dass $L \subseteq L(M)$. Außerdem können wir leicht prüfen, dass auch keine anderen Wörter akzeptiert werden. Das sind nämlich alle Wörter, die auf 1 enden, alle, die auf 10 enden, sowie das Wort $w = 0$:

$$\begin{aligned}\hat{\delta}(z_0, u1) &= \delta(\hat{\delta}(z_0, u), 1) = \delta(z_i, 1), \text{ wobei } 0 \le i \le 2\\ &= z_0 \notin F\\ \hat{\delta}(z_0, u10) &= \delta(\hat{\delta}(z_0, u1), 0) = \delta(z_0, 0) = z_1 \notin F\\ \delta(z_0, 0) &= z_1 \notin F\end{aligned}$$

Somit gilt $L = L(M)$, und wir wissen, dass es sich bei $L = \{w \in \Sigma^* | w = u00 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$ um eine reguläre Sprache handelt.

Beginnen wir diesmal mit dem Automaten M, der wieder über dem Alphabet $\Sigma = \{0, 1\}$ arbeitet:



Welche Sprache akzeptiert dieser Automat? Bei genauerem Hinsehen stellt sich heraus, dass man mindestens drei aufeinanderfolgende Nullen benötigt, um vom Startzustand z_0 in den Endzustand z_3 zu gelangen. Ist man einmal im Endzustand, kann dieser nicht mehr verlassen werden.

Tatsächlich bilden alle Wörter, die drei aufeinanderfolgende Nullen enthalten, die von M akzeptierte Sprache:

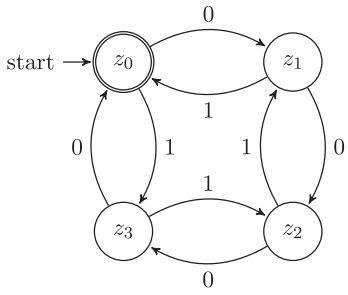
$L(M) = \{w \in \Sigma^* | w = u000v \text{ mit } u, v \in \Sigma^* \text{ beliebig.}\}$

Zum Beweis rechnen wir wieder als Erstes nach, dass alle Wörter der Sprache akzeptiert werden:

$$\begin{aligned}\hat{\delta}(z_0, u000v) &= \hat{\delta}(\hat{\delta}(z_0, u), 000v) \\ &= \hat{\delta}(z_i, 000v), 0 \leq i \leq 3 \\ &= \hat{\delta}(\hat{\delta}(z_i, 000), v)) \\ &= \hat{\delta}(z_3, v) = z_3 \in F\end{aligned}$$

Auf der anderen Seite gibt es keine andere Möglichkeit, in den Endzustand zu gelangen, als mit Hilfe dreier aufeinanderfolgender Nullen. Es werden also keine Wörter akzeptiert, die nicht in $L(M)$ liegen.

Schließlich möchte ich Ihnen einen Automaten über dem Alphabet $\Sigma = \{0, 1\}$ zeigen, der gewissermaßen rechnen kann:



Gibt man probeweise einige Wörter in den Automaten ein, zeigt sich folgendes Verhalten: Die Nullen im Eingabewort sorgen dafür, dass man sich im Uhrzeigersinn durch den Automaten bewegt: Nach vier Nullen ist man wieder im Startzustand, der gleichzeitig auch Endzustand ist. Die Einsen sorgen für eine gegenläufige Bewegung. Beispielweise ist $\hat{\delta}(z_0, 001) = z_1$ und $\hat{\delta}(z_0, 100100) = z_2$. Offenbar ist also die Differenz zwischen der Anzahl der Nullen und Einsen im Eingabewort entscheidend dafür, in welchem Zustand man am Ende landet. Stellt diese Differenz ein Vielfaches von 4 dar, hat man den Automaten mehrfach umrundet, und das Wort wird akzeptiert.

Etwas präziser formuliert: Ist $N_0(w)$ bzw. $N_1(w)$ die Anzahl der Nullen bzw. Einsen in einem Eingabewort w, so gilt:

$$L(M) = \{w \in \Sigma^* | N_0(w) - N_1(w) \equiv 0 \mod 4\}$$

Der Automat kann also prüfen, ob die Differenz der Anzahl der Nullen und Einsen im Eingabewort ein Vielfaches von 4 darstellt. Ist man an einem anderen Rest beim Teilen durch 4 interessiert, kann man einfach den Endzustand verändern: Erklärt man z. B. z_3 anstelle von z_0 zum Endzustand, so erhält man natürlich die Sprache

$$L(M) = \{w \in \Sigma^* | N_0(w) - N_1(w) \equiv 3 \mod 4\}$$

Diese Beispiele zeigen, dass es durchaus eine große Vielfalt regulärer Sprachen gibt. Können wir irgendwelche allgemeinen Aussagen darüber treffen, welche Sprachen regulär sind? Wie steht es beispielsweise mit Sprachen, die nur aus einer endlichen Zahl von Wörtern bestehen? Für solche Sprachen lässt sich tatsächlich immer ein passender DFA finden.

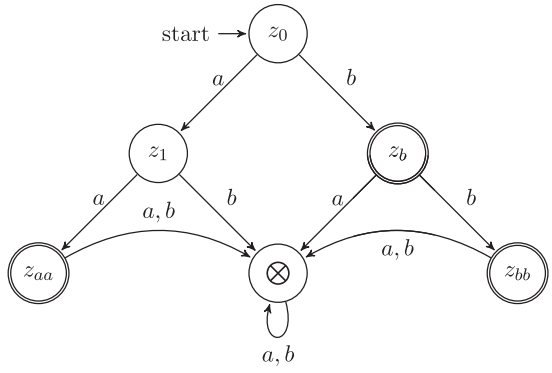
Endliche Sprachen sind regulär.

Σ sei ein Alphabet. Dann ist jede endliche Teilmenge

$L = \{w_1, w_2, \ldots, w_n\} \subset \Sigma^*$ eine reguläre Sprache.

Zum Beweis dieser Aussage gehen wir ohne Beschränkung der Allgemeinheit davon aus, dass $\Sigma = \{a, b\}$ ist. Wir konstruieren einen zu L passenden DFA M, indem wir seinen Übergangsgraphen im Prinzip als Binärbaum anlegen mit dem Startzustand z_0 als Wurzelknoten. Für jedes der Wörter $w \in L$ reservieren wir einen eigenen Endzustand $z_w \in F$ als Blattknoten. Wir konstruieren den Baum so, dass der Weg von der Wurzel zu einem Endzustand z_w der Abfolge der Symbole in w entspricht. Da es nur endliche viele Wörter in L gibt, brauchen wir dafür auch nur endlich viele Zustände. Für ungültige Symbolfolgen, also Teilwörter, die keine Chance mehr haben, zu einem der Wörter in L zu werden, stellen wir noch zusätzlich einen Fehlerzustand bereit, der nicht mehr verlassen werden kann und den wir mit einem ⊗ markieren. Leider zerstört die Notwendigkeit des Fehlerzustands die schöne Baumstruktur des Übergangsgraphen. An der Gültigkeit der Konstruktion ändert dies jedoch nichts. Schauen wir uns ein Beispiel an:

$\Sigma = \{a, b\}, L = \{b, aa, bb\}$. Der zu dieser Sprache passende DFA hat drei Endzustände z_b, z_{aa}, z_{bb} und den folgenden Übergangsgraphen:



Das Pumping Lemma

Gibt es eigentlich auch nicht-reguläre Sprachen? Die Sprache

$$L = \{w \in \Sigma^* | w = a^n b^n, n \geq 0\}$$

über dem Alphabet $\Sigma = \{a, b\}$ bildet ein klassisches Beispiel einer nicht-regulären Sprache: So lange n einen festen Wert besitzt, ist die Sprache L endlich und somit regulär. Läßt man jedoch alle Werte für n zu, bräuchte ein Automat, der L akzeptiert, eine Möglichkeit, sich zu merken, wie viele a das Eingabewort enthält. Die einzige Speichermöglichkeit für einen DFA besteht jedoch in seinen Zuständen. Kann n beliebig groß werden, braucht man also auch beliebig viele Zustände zum Speichern. Der Automat kann also nicht endlich sein.

Tatsächlich kann man einer Sprache in vielen Fällen ansehen, dass sie nicht-regulär ist. Reguläre Sprachen müssen notwendigerweise bestimmte Eigenschaften erfüllen, und wenn eine Sprache diese Eigenschaften nicht besitzt, kann sie auch nicht-regulär sein.

Stellen wir uns einen DFA M mit N Zuständen und akzeptierter Sprache $L(M)$ vor. Angenommen, ein Wort $x \in L(M)$ hat N oder mehr Symbole: Dann muss beim Abarbeiten des Worts x mindestens ein Zustand z von M mehrfach besucht werden (so genanntes *Schubfachprinzip* bzw. im Englischen auch *pigeonhole principle*: Wenn n Tauben k Löcher anfliegen, wobei $n > k$, dann gibt es mindestens ein Loch, dass von mehreren Tauben besucht wird).

Das Wort $x \in L(M)$ zerfällt also in einen ersten Bestandteil u mit $\hat{\delta}(z_0, u) = z$, einen mittleren Bestandteil v mit $\hat{\delta}(z, v) = z$ und einen End-Bestandteil w mit $\hat{\delta}(z, w) = z_e \in F$. Im Übergangsgraphen von M sieht das ungefähr so aus:

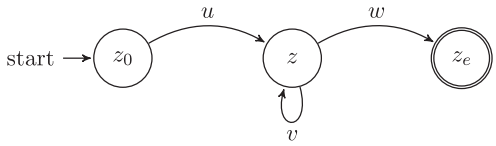


Abbildung 1.5: Verarbeitung eines langen Worts x im DFA

Insgesamt gilt nach Abb. 1.5 also $x = uvw$, wobei v für einen Kreis durch den Übergangsgraphen sorgt, der bei z beginnt und endet. Dabei ist klar, dass der Kreis-Bestandteil v auch weggelassen werden kann, ohne dass wir aus $L(M)$ herausfallen: Das Restwort uw führt immer noch von z_0 nach z_e, also ist $uw \in L(M)$.

Mehr noch: Der Kreis v kann auch mehrfach durchlaufen oder auch *gepumpt* werden, ohne dass das Wort aus $L(M)$ herausfällt. Beide Eigenschaften lassen sich in der Form

$$uv^i w \in L(M) \text{ für } i \geq 0$$

zusammenfassen. Diese Überlegung bildet den Kern des Beweises für das so genannte *Pumping Lemma* (mit *Lemma* bezeichnet man in der Mathematik eine wichtige Hilfsaussage):

Pumping Lemma

L sei eine reguläre Sprache. Dann gibt es eine Zahl $N \in \mathbb{N}$, sodass sich jedes Wort $x \in L$ mit $|x| \geq N$ in der Form $x = uvw$ schreiben lässt, wobei für die Bestandteile u, v, w gilt:

1. $|uv| \leq N$
2. $|v| \geq 1$
3. $\forall i \geq 0 : uv^i w \in L$

Die zweite und dritte Bedingung ergeben sich direkt aus unseren obigen Überlegungen. Die erste Bedingung $|uv| \leq N$ ergibt sich aus der Tatsache, dass wir bei Eingabe von N Zeichen in den Automaten genau $(N + 1)$mal einen Zustand besuchen. Da der Automat nur N Zustände besitzt, muss also ein Zustand mehrfach besucht werden, und das bedeutet, dass der Schleifenbestandteil v und sein Vorgängerbestandteil u beide innerhalb der ersten N Zeichen liegen müssen.

Jedes hinreichend lange Wort in L muss also einen Bestandteil v enthalten, der sich beliebig oft wiederholen (»aufpumpen«) lässt, ohne dass man die Sprache verlässt. Das Pumping Lemma bildet somit ein **notwendiges** Kriterium dafür, dass eine formale Sprache regulär ist.

Betrachten wir z.B. die Sprache $L = \{w \in \Sigma^* | w = u00 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$ über dem Alphabet $\Sigma = \{0, 1\}$. Wir wissen bereits, dass diese Sprache regulär ist, und tatsächlich erfüllt sie die Bedingungen des Pumping Lemma: Alle Wörter $x \in L$ mit $|x| \geq 3$ besitzen einen pumpbaren Bestandteil v, nämlich das erste Symbol von x. Dieses kann weggelassen oder beliebig wiederholt werden, ohne dass sich an der Zugehörigkeit zu L etwas ändert (als u nehmen wir einfach das leere Wort, somit ist $|uv| = 1 \leq 3$, und w ist der Rest des Worts nach dem ersten Symbol).

Wie steht es mit endlichen Sprachen? Wir wissen, dass alle endlichen Sprachen regulär sind, aber eine endliche Aufzählung von Wörtern enthält doch sicher keine pumpbaren Bestandteile? Nein, das nicht, aber im Pumping Lemma ist auch nur von Wörtern die Rede, die eine gewisse Mindestlänge N überschreiten (und diese Mindestlänge dürfen wir selbst festlegen). Wählen wir also N größer als die größte vorkommende Wortlänge in L, dann gibt es gar keine Wörter $x \in L$ mit $|x| \geq N$, so dass sich die Prüfung der drei Bedingungen im Pumping Lemma erübrigt. Nur unendliche reguläre Sprachen müssen also pumpbare Bestandteile enthalten.

Das Pumping Lemma wird in erster Linie dazu genutzt, um zu zeigen, dass eine Sprache **nicht** regulär ist: Wie wir gesehen haben, muß eine reguläre Sprache das Pumping Lemma erfüllen. Enthält also eine Sprache keine pumpbaren Bestandteile, kann sie auch nichtregulär sein. Dieses Beweisverfahren wird in der formalen Logik **Kontraposition** genannt: Das Pumping Lemma stellt eine Aussage der Form $A \Rightarrow B$ dar. Dazu ist die Aussage $\neg B \Rightarrow \neg A$ logisch äquivalent (siehe auch Kapitel 12). Da das Pumping Lemma praktisch ausschließlich

in dieser Form angewendet wird, formulieren wir die Kontraposition des Pumping Lemma noch einmal explizit:

Kontraposition des Pumping Lemma

L sei eine formale Sprache. Gibt es **keine** Zahl $N \in \mathbb{N}$, so dass sich **jedes** Wort $x \in L$ mit $|x| \geq N$ in der Form $x = uvw$ schreiben lässt, wobei für die Bestandteile u, v, w gilt:

1. $|uv| \leq N$

2. $|v| \geq 1$

3. $\forall i \geq 0 : uv^i w \in L$

so ist L **nicht** regulär.

Benutzen wir also das Pumping Lemma in der Kontrapositionsform, um zu zeigen, dass eine Sprache L nicht-regulär ist. Konkret gehen wir dabei von einer Mindestlänge N aus. Gelingt es uns dann, zu jedem N auch nur ein Wort $x \in L$ mit $|x| \geq N$ zu finden, für das die drei Eigenschaften aus dem Pumping Lemma nicht erfüllt sind, kann L nicht-regulär sein.

$\Sigma = \{a, b\}, L = \{x \in \Sigma^* | x = a^n b^n, n \geq 0\}$.

Für diese Sprache haben wir uns bereits weiter oben überlegt, dass sie nicht-regulär sein kann. Mit dem Pumping Lemma können wir nun einen formalen Beweis dafür angeben, indem wir annehmen, es gäbe eine Zahl N, so dass jedes Wort $x \in L$ mit $|x| \geq N$ die drei Bedingungen aus dem Pumping Lemma erfüllt.

Betrachten wir speziell das Wort $x = a^N b^N$. Offenbar ist $x \in L$ und $|x| = 2N \geq N$, so dass sich nach dem Pumping Lemma x zerlegen lassen muss:

$x = uvw$ mit $|uv| \leq N$ und $|v| \geq 1$.

Da die ersten N Symbole von x lauter a's sind, bestehen auch die Teilwörter uv und v aus lauter a's. Wir können also schreiben:

$v = a^{|v|}$ und $x = uvw = ua^{|v|}w$, wobei $|v| \geq 1$. Nach der dritten Bedingung des Pumping Lemma muß aber z.B. auch das Wort $uv^2w = ua^{2|v|}w \in L$ sein. uv^2w enthält $(N + |v|)$ a's und N b's, da sich durch das Verdoppeln von v an der Anzahl der b's nichts geändert hat. Also ist $uv^2w \notin L$. Widerspruch!

Es gibt also keine Zahl N mit den gewünschten Eigenschaften, und L ist damit nicht-regulär.

$\Sigma = \{1\}, L = \{x \in \Sigma^* | x = 1^{i^2} \text{ für } i \geq 0\}$.

Wir betrachten also lange Listen von Einsen und würden gerne mit einem DFA prüfen, ob die Länge der Liste eine Quadratzahl darstellt. Leider zeigt uns das Pumping Lemma, dass es einen solchen DFA nicht geben kann. Wir gehen

wieder davon aus, dass die Sprache regulär ist und dass die drei Bedingungen des Pumping Lemma für eine bestimmte Zahl N gelten. Betrachten wir speziell das Wort $x = 1^{N^2} \in L$, dann ist $|x| = N^2 \geq N$ und nach dem Pumping Lemma muß gelten: $x = 1^{N^2} = uvw$ mit $|uv| \leq N, |v| \geq 1$ und $uv^i w \in L$ für $i \geq 0$. Insbesondere ist also auch $uv^2w \in L$. Wegen $|v| \geq 1$ gilt jetzt:

$N^2 = |x| = |uvw| < |uv^2w|.$

Und wegen $|uv| \leq N$ gilt:

$|uv^2w| = |uvwv| = |uvw| + |v| = N^2 + |v| \leq N^2 + N.$

Wir haben also die Länge von uv^2w zwischen zwei Zahlen einschränken können:

$N^2 < |uv^2w| \leq N^2 + N.$

Aber weil $N^2 + N < N^2 + 2N + 1 = (N + 1)^2$, gilt

$N^2 < |uv^2w| < (N + 1)^2$

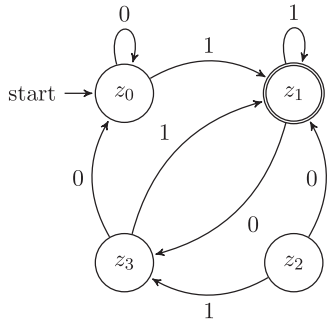
Deshalb kann $|uv^2w|$ selbst keine Quadratzahl sein und das Wort uv^2w liegt nicht in L. Widerspruch!

Minimalautomaten

Hat man eine reguläre Sprache L, so ist der L akzeptierende DFA M keineswegs eindeutig. Es lassen sich immer verschiedene Automaten konstruieren, die die gleiche Sprache akzeptieren. Alle diese Automaten nennen wir **äquivalent**. Unter allen äquivalenten Automaten ist natürlich derjenige am interessantesten, der beim Erkennen von L mit den wenigsten Zuständen auskommt: Er arbeitet in gewisser Weise am *effizientesten* von allen äquivalenten Automaten (während alle äquivalenten Automaten *effektiv* sind in dem Sinn, dass sie alle L akzeptieren).

L sei formale Sprache über dem Alphabet Σ und M ein DFA, der L akzeptiert. Dann heißt M **Minimalautomat** für L, wenn es keinen zu M äquivalenten DFA gibt, der weniger Zustände hat als M.

Betrachten wir den folgenden DFA M über dem Alphabet $\Sigma = \{0, 1\}$:

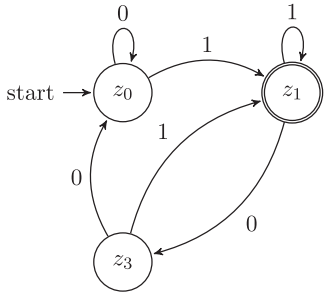


Welche Sprache akzeptiert er? Der Endzustand kann nur über eine 1 erreicht werden. Akzeptierte Wörter müssen also auf 1 enden. Auf der anderen Seite werden aber auch alle Wörter, die auf 1 enden, akzeptiert:

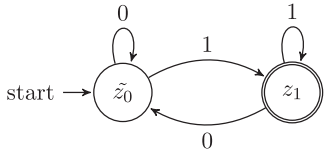
$$\hat{\delta}(z_0, u1) = \delta(\hat{\delta}(z_0, u), 1) = \delta(z_i, 1) \text{ mit } i \in \{0, 1, 3\}$$
$$= z_1 \in F$$

Damit ist klar, dass $L(M) = \{w \in \Sigma^* | w = u1 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$. Woher aber wußten wir, dass $\hat{\delta}(z_0, u)$ niemals z_2 sein kann? Weil von z_2 nur Pfeile ausgehen, aber keine Pfeile in z_2 enden, kann z_2 niemals angenommen werden. Man nennt solche Zustände **nicht erreichbar**.

Offenbar ändert es nichts an der akzeptierten Sprache, wenn man nicht erreichbare Zustände einfach weglässt. Wir erhalten einen kleineren, zu M äquivalenten Automaten:



Ist das nun der Minimalautomat für unsere Sprache $L = \{w \in \Sigma^* | w = u1 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$? Nein, natürlich geht es noch etwas kleiner:



Hier wurden die Zustände z_0 und z_3 zu einem einzigen Zustand $\tilde{z}_0$ zusammengelegt. Man kann dies immer dann tun, wenn zwei Zustände in einem gewissen Sinne *äquivalent* sind, den wir gleich exakt definieren werden.

$M = (Z, \Sigma, \delta, z_0, F)$ sei ein DFA.

- ✔ Ein Zustand $z \in Z$ mit der Eigenschaft $\forall u \in \Sigma^* : \hat{\delta}(z_0, u) \neq z$ heißt **nicht erreichbar.**
- ✔ Nun sei $z \in Z$ ein erreichbarer Zustand. Mit $L(M, z)$ bezeichnen wir die Sprache, die M akzeptieren würde, wenn z anstelle von z_0 der Startzustand wäre. Wir definieren eine Äquivalenzrelation R_M auf der Zustandsmenge Z via

  $$(z_1, z_2) \in R_M \Leftrightarrow L(M, z_1) = L(M, z_2)$$

Im Beispiel oben ist $L(M, z_0) = L(M, z_3) = \{w \in \Sigma^* | w = u1 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$. Wie genau funktioniert nun das Zusammenlegen äquivalenter Zustände? Wie üblich bezeichnen wir mit $[z]$ die Menge der zu $z \in Z$ äquivalenten Zustände, also die Äquivalenzklasse von z. Die äquivalenten Zustände in $[z]$ werden zu einem neuen Zustand $\tilde{z}$ zusammengelegt. Ziel ist es, alle Zustände in $[z]$ durch den neuen Zustand $\tilde{z}$ zu ersetzen, ohne dass sich etwas an der von M akzeptierten Sprache ändert.

Dazu müssen wir insbesondere für den neuen Zustand $\tilde{z}$ die Funktionswerte $\delta(\tilde{z}, a)$ definieren. Da $\tilde{z} = [z]$ aus mehreren Zuständen bestehen kann, schauen wir uns dazu im Übergangsdiagramm von M an, in welche Zustände die Pfeile führen, die von den äquivalenten Zuständen z_i in $[z]$ ausgehen. Um keine Informationen zu verlieren, vereinigen wir vorläufig alle diese von $[z]$ aus erreichten Zustände in einer Menge:

$$\forall a \in \Sigma : \delta(\tilde{z}, a) = \bigcup_{z_i \in [z]} \delta(z_i, a)$$

Im Beispiel oben gilt

$$\delta(\tilde{z_0}, 0) = \bigcup_{z_i \in [z_0]} \delta(z_i, 0) = \delta(z_0, 0) \cup \delta(z_3, 0) = \{z_0\} \text{ und}$$

$$\delta(\tilde{z_0}, 1) = \bigcup_{z_i \in [z_0]} \delta(z_i, 1) = \delta(z_0, 1) \cup \delta(z_3, 1) = \{z_1\}.$$

Die so erhaltenen Mengen von Zuständen identifizieren wir mit den Äquivalenzklassen ihrer Elemente.

Aber was passiert, wenn bei der Vereinigung Mengen aus mehr als einem Zustand entstehen? Zum Glück sorgt dann die Äquivalenz der Zustände in $[z]$ dafür, dass auch die Zustände $y \in \bigcup_{z_i \in [z]} \delta(z_i, a)$ zueinander äquivalent sind. Es gilt nämlich die folgende Hilfsaussage:

Sind z_1 und z_2 äquivalente Zustände, so sind auch $r = \delta(z_1, a)$ und $s = \delta(z_2, a)$ für alle $a \in \Sigma$ äquivalent.

Weil z_1 und z_2 äquivalent sind, gilt für jedes $w \in \Sigma^*$:

$$\hat{\delta}(z_1, w) \in F \Leftrightarrow \hat{\delta}(z_2, w) \in F.$$

Ist nun $w = au$ für ein Symbol $a \in \Sigma$, folgt daraus

$$\hat{\delta}(z_1, au) = \hat{\delta}(\delta(z_1, a), u) \in F \Leftrightarrow \hat{\delta}(\delta(z_2, a), u) \in F.$$

Also sind $r = \delta(z_1, a)$ und $s = \delta(z_2, a)$ äquivalent.

Folglich sind auch alle $y \in \bigcup_{z_i \in [z]} \delta(z_i, a)$ zueinander äquivalent: Egal welches y man auswählt, man erhält am Ende immer die gleiche Äquivalenzklasse. Bestimmt man also für alle

Zustände $z \in Z$ ihre Äquivalenzklassen und definiert dadurch neue Zustände $\tilde{z}$, so erhalten wir wieder eine wohldefinierte Übergangsfunktion δ durch

$$\forall a \in \Sigma : \delta(\tilde{z}, a) = \tilde{y}, \text{ wobei } y \in \bigcup_{z_i \in [z]} \delta(z_i, a)$$

Wir überlegen uns noch, dass der Automat mit den neuen Zuständen die gleiche Sprache wie M akzeptiert. Dazu brauchen wir zunächst eine Menge $\tilde{F}$ von Endzuständen: Wir setzen

$$\tilde{F} = \{\tilde{z}_i | z_i \in F\}$$

und betrachten ein beliebiges Wort $w = a_1 a_2 \dots a_n \in \Sigma^*$. Dann gilt: $x \in L(M) \Leftrightarrow \hat{\delta}(z_0, x) \in F$, was bedeutet, dass eine Folge von Zuständen $z_0, z_1, \dots, z_n$ existiert mit

$$\delta(z_0, a_1) = z_1, \delta(z_1, a_2) = z_2, \dots, \delta(z_{n-1}, a_n) = z_n \in F.$$

All dies spielt sich im ursprünglichen Automaten M ab. Können wir diese Folge irgendwie auf die neuen Zustände $\tilde{z}$ übertragen? Wegen $\delta(z_0, a_1) = z_1$ und $z_0 \in [z_0]$ ist auf jeden Fall $z_1 \in \bigcup_{z_i \in [z_0]} \delta(z_i, a_1)$. Da es auf den Repräsentanten der Äquivalenzklasse nicht ankommt, dürfen wir schließen:

$$\delta(\tilde{z}_0, a_1) = \tilde{z}_1,$$

und die gleiche Schlussweise funktioniert natürlich auch für die anderen Zustandsübergänge der Folge. Am Ende steht der Übergang

$$\delta(\tilde{z}_{n-1}, a_n) = \tilde{z}_n \in \tilde{F}, \text{ weil } z_n \in F.$$

$x \in L(M)$ wird also auch von dem neuen Automaten akzeptiert.

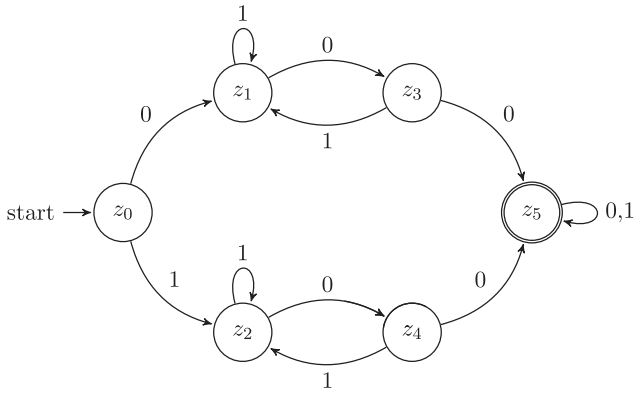
Die beiden Reduktionsschritte »Weglassen nicht erreichbarer Zustände« und »Zusammenlegen äquivalenter Zustände« reichen aus, um aus jedem Automaten den Minimalautomaten zu destillieren:

Konstruktion des Minimalautomaten

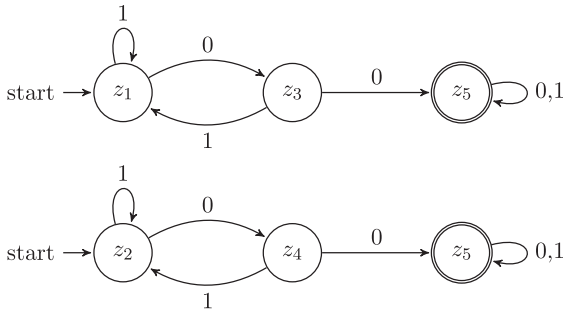
$M = (Z, \Sigma, \delta, z_0, F)$ sei ein DFA und L die von ihm akzeptierte Sprache.

Der Automat $\tilde{M}$, der aus M durch Weglassen nicht erreichbarer Zustände und Zusammenlegen äquivalenter Zustände entsteht, ist der Minimalautomat für L.

Betrachten wir zur Gewöhnung an die obigen Begriffe den folgenden, etwas größeren DFA:



Er enthält keine nicht erreichbaren Zustände, aber z_1 und z_2 scheinen sich in ihrem Verhalten sehr ähnlich zu sein. Tatsächlich sind sie äquivalent, wie man sofort erkennt, wenn man z_1 bzw. z_2 zum Startzustand erklärt. Dann werden nämlich plötzlich ganze drei Zustände nicht erreichbar. Entfernt man sie, erhält man die folgenden beiden Automaten:



Aus diesen beiden Automaten lesen wir ab, dass

$$L(M, z_1) = L(M, z_2) = \{w \in \Sigma^* | w = u00v \text{ mit } u, v \in \Sigma^* \text{ beliebig.}\}.$$

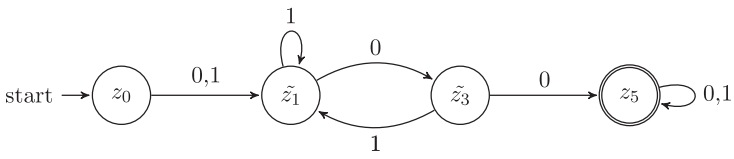
Außerdem zeigen uns die beiden Automaten, dass auch noch z_3 und z_4 äquivalent sind. Die entsprechende Sprache ist etwas komplizierter:

$$L(M, z_3) = L(M, z_4) = \{w \in \Sigma^* | w = (1^n 0)^m 0v \text{ mit } m \geq 0, n \geq 1 \text{ und } v \in \Sigma^* \text{ beliebig.}\}.$$

Wir legen die äquivalenten Zustände z_1, z_2 zu $\tilde{z}_1$ und z_3, z_4 zu $\tilde{z}_3$ zusammen. Dann gilt zum Beispiel:

$$\delta(\tilde{z}_1, 0) = \bigcup_{z \in [z_1]} \delta(z, 0) = \{z_3, z_4\} = [z_3] = \tilde{z}_3$$

Insgesamt ergibt sich dieser Minimalautomat:



Die akzeptierte Sprache lässt sich von diesem Automaten deutlich einfacher ablesen. Sie lautet:

$L(M) = \{w \in \Sigma^* | w = (0|1)u00v \text{ mit } u, v \in \Sigma^* \text{ beliebig.}\}$

Zum Finden äquivalenter Zustände sind wir nicht auf genaues Hinsehen angewiesen. Man kann auch einen Algorithmus angeben, der äquivalente Zustände findet. Der Algorithmus basiert auf zwei Beobachtungen:

- ✔ Zwei Zustände $z_e \in F$ und $z \notin F$ können nicht äquivalent sein, da $L(M, z_e)$ das leere Wort enthält, $L(M, z)$ aber nicht.
- ✔ Sind zwei Zustände $r = \delta(z_1, a)$ und $s = \delta(z_2, a)$ für ein Symbol $a \in \Sigma$ nicht äquivalent, dann können auch z_1 und z_2 nicht äquivalent sein.

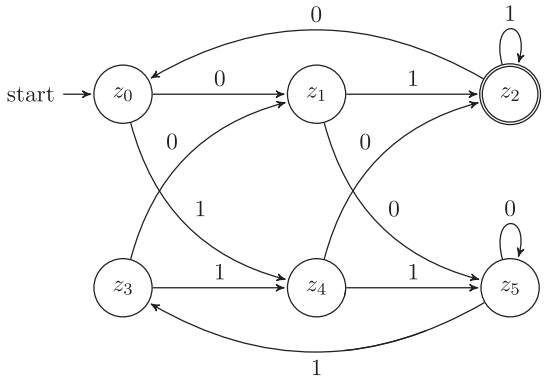
Die zweite Beobachtung ist die Kontraposition der folgenden, leichter einzusehenden Aussage, die wir oben bereits bei der Konstruktion des Minimalautomaten genutzt haben:

Sind z_1 und z_2 äquivalente Zustände, so sind auch $r = \delta(z_1, a)$ und $s = \delta(z_2, a)$ für alle $a \in \Sigma$ äquivalent.

Die zweite Beobachtung lässt sich weiter verallgemeinern zu

- ✔ Sind zwei Zustände $r = \hat{\delta}(z_1, w)$ und $s = \hat{\delta}(z_2, w)$ für ein Wort $w \in \Sigma^*$ nicht äquivalent, dann können auch z_1 und z_2 nicht äquivalent sein.

Damit kann man für alle Zustandspaare (z_1, z_2) (bei N Zuständen sind das genau $\frac{N(N-1)}{2}$ Stück) bestimmen, ob sie äquivalent sind, indem man mit der ersten Beobachtung startet und von den damit gefundenen, nicht äquivalenten Zustandsparen ausgehend, mit Hilfe der zweiten Beobachtung weitere, nicht äquivalente Zustandspaare bestimmt. Anstatt den Algorithmus zu formalisieren und zu beweisen, dass man auf diese Weise tatsächlich alle Zustandspaare erwischt, schauen wir uns lieber einen Beispielautomaten an:



Nicht äquivalent sind auf jeden Fall die Zustandspaare (z_0, z_2), (z_1, z_2), (z_2, z_3), (z_2, z_4) und (z_2, z_5), da diese jeweils einen Endzustand und einen Nicht-Endzustand enthalten. Als Nächstes betrachten wir die Pfeile, mit denen der Endzustand z_2 erreicht werden kann: Es ist

$$\delta(z_4, 0) = z_2 \in F \text{ und } \delta(z_1, 1) = z_2 \in F.$$

Wegen der zweiten Beobachtung

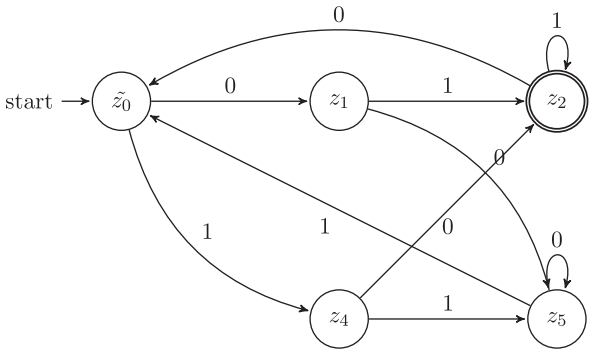
$r = \delta(y, a)$ und $s = \delta(z, a)$ nicht äquivalent $\Rightarrow$ y und z nicht äquivalent

liefert uns jeder Nicht-Endzustand $s = \delta(z, 0)$, der über eine 0 erreicht werden kann, einen weiteren Zustand z, der nicht äquivalent zu $y = z_4$ ist. Analog liefert uns jeder Nicht-Endzustand $s = \delta(z, 1)$, der über eine 1 erreicht werden kann, einen Zustand z, der nicht äquivalent zu $y = z_1$ sein kann:

$$\begin{aligned}
\delta(z_0, 0) &= z_1 \Rightarrow z_0 \not\equiv z_4, \ \delta(z_1, 0) = z_5 \Rightarrow z_1 \not\equiv z_4, \\
\delta(z_3, 0) &= z_1 \Rightarrow z_3 \not\equiv z_4, \ \delta(z_5, 0) = z_5 \Rightarrow z_5 \not\equiv z_4, \\
\delta(z_0, 1) &= z_4 \Rightarrow z_0 \not\equiv z_1, \ \delta(z_3, 1) = z_4 \Rightarrow z_3 \not\equiv z_1, \\
\delta(z_5, 1) &= z_3 \Rightarrow z_5 \not\equiv z_1
\end{aligned}$$

Damit haben wir bereits 12 von 15 möglichen Zustandspaaren als nicht äquivalent erkannt. Es bleiben noch die Paare (z_0, z_3), (z_0, z_5) und (z_3, z_5).

Weil $\hat{\delta}(z_0, 01) = z_2$ und $\hat{\delta}(z_5, 01) = z_3$ gilt, wobei z_2 und z_3 nicht äquivalent sind, können auch z_0 und z_5 nicht äquivalent sein. Ebenso sind wegen $\hat{\delta}(z_3, 01) = z_2$ und $\hat{\delta}(z_5, 01) = z_3$ die Zustände z_3 und z_5 nicht äquivalent. Es bleibt als einziges äquivalentes Zustandspaar das Paar (z_0, z_3). Der dazugehörige Minimalautomat hat dieses Aussehen:



Der Satz von Myhill und Nerode

Leider ist das Pumping Lemma kein hinreichendes Kriterium für die Regularität einer Sprache, d. h., auch wenn eine Sprache L die Bedingungen des Pumping Lemma erfüllt, kann es manchmal sein, dass L nicht-regulär ist. Wir werden uns weiter unten ein Beispiel für eine solche Sprache anschauen. Zunächst möchte ich Ihnen aber ein notwendiges und hinreichendes Kriterium für Regularität vorstellen.

Wir haben etwas weiter oben eine Äquivalenzrelation R_M für die Zustände eines DFAs M eingeführt: Zwei Zustände z_1 und z_2 sind äquivalent, falls M die gleiche Sprache $L(M, z_1) = L(M, z_2)$ akzeptiert, egal ob man z_1 oder z_2 zum Startzustand erklärt. Die Relation R_M lässt sich auf Wörter $\in \Sigma^*$ übertragen: Sind nämlich x und y zwei Wörter, die z_0 in äquivalente Zustände z_1, z_2 überführen und hängt man dann einen weiteren Bestandteil $z \in L(M, z_1)$ an x bzw. y an, so sind beide Ergebnisse xz und yz akzeptierte Wörter $\in L(M)$. Ist $z \notin L(M, z_1)$, so sind xz und yz auch beide nicht in $L(M)$.

Wir identifizieren deshalb die Wörter x und y mit den über sie erreichten Zuständen z_1 und z_2 und erklären zwei Wörter $x, y \in \Sigma^*$ für äquivalent bezogen auf eine Sprache L, *wenn sie sich gleich verhalten, und zwar bezogen auf das Anhängen anderer Wörter und Mitgliedschaft des Ergebnisses in L.* Äquivalente Wörter werden in Teilmengen von Σ^*, den Äquivalenzklassen von R_L, zusammengefasst, und die Anzahl der Äquivalenzklassen heißt **Index** von L.

L sei formale Sprache über dem Alphabet Σ und $x, y \in \Sigma^*$.

Durch

$$(x, y) \in R_L \Leftrightarrow \forall z \in \Sigma^* : (xz \in L \Leftrightarrow yz \in L)$$

wird eine Äquivalenzrelation in Σ^* definiert. Die Anzahl der Äquivalenzklassen von R_L heißt **Index** von L.

Mit diesem Begriff gilt jetzt:

Satz von Myhill und Nerode

Eine Sprache L ist genau dann regulär, wenn ihr Index endlich ist.

Wir werden den Satz von Myhill und Nerode weiter unten beweisen und im Zuge dessen die Bedeutung des Index einer Sprache genau verstehen. Zunächst sollen aber einige Beispiele die Leistungsfähigkeit des Satzes verdeutlichen.

$\Sigma = \{0, 1\}$. $L = \{w \in \Sigma^* | w = u00 \text{ mit } u \in \Sigma^* \text{ beliebig.}\}$.

Wir wissen bereits, dass diese Sprache regulär ist, da wir weiter oben den dazu passenden DFA untersucht haben. Wir gehen nun den Weg über den Satz von Myhill / Nerode und berechnen den Index der Sprache. Nach dem Satz von Myhill und Nerode sollte er endlich sein. Wir beginnen mit dem leeren Wort ϵ und fragen uns, welche Wörter zu ϵ äquivalent sind, d.h. welche Wörter sich in Bezug auf das Anhängen weiterer Wortteile und die Mitgliedschaft in L gleich wie ϵ verhalten. Das Verhalten von ϵ ist schnell geklärt:

$\epsilon z \in L \Leftrightarrow z \in L$. So verhalten sich genau die Wörter $y \in \Sigma^*$, die auf 1 enden. Somit lautet die Äquivalenzklasse von ϵ:

$[\epsilon] = \{y \in \Sigma^* | y = u1 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$.

Betrachten wir nun die Äquivalenzklasse des Worts 0:

$0z \in L \Leftrightarrow z \in L$ **oder** $z = 0$. So verhalten sich außer der 0 genau die Wörter $y \in \Sigma^*$, die auf 10 enden. Somit lautet die Äquivalenzklasse von 0:

$[0] = \{y \in \Sigma^* | y = u10 \text{ mit } u \in \Sigma^* \text{ beliebig}\} \cup \{0\}$.

Als Letztes betrachten wir das Verhalten des Worts 00:

$00z \in L \Leftrightarrow z \in L$ **oder** $z = \epsilon$. So verhalten sich genau die Wörter $y \in \Sigma^*$, die auf 00 enden. Somit lautet die Äquivalenzklasse von 00:

$[00] = \{y \in \Sigma^* | y = u00 \text{ mit } u \in \Sigma^* \text{ beliebig}\}$.

Damit sind alle Äquivalenzklassen bestimmt und der Index von L ist 3. Wie können wir uns sicher sein, dass uns nicht doch noch eine Äquivalenzklasse fehlt? Die Äquivalenzklassen einer Äquivalenzrelation über einer Menge M bilden eine **Partition** von M, was unter anderem heißt, dass sich jedes Element von M in einer der Äquivalenzklassen wiederfindet. Hier ist $M = \{0, 1\}^*$, und wir müssen prüfen, ob sich jeder binäre String in einen der drei Äquivalenzklassen einordnet. Das ist aber offenbar der Fall, da jeder binäre String entweder mit 1, 10 oder 00 endet.

$\Sigma = \{a, b\}, L = \{x \in \Sigma^* | N_a(x) = N_b(x)\}$.

In dieser Sprache finden sich alle Strings, in denen die Anzahl der *a*'s gleich der Anzahl der *b*'s ist. Wir haben es also mit einer Obermenge der Sprache $L = \{x \in \Sigma^* | x = a^n b^n, n \geq 0\}$. zu tun, von der wir bereits wissen, dass sie nicht-regulär ist. Wir bestimmen wieder die Äquivalenzklassen on R_L, beginnend beim leeren Wort:

$\epsilon z \in L \Leftrightarrow z \in L$. So verhalten sich genau die Wörter $y \in \Sigma^*$, die schon in L sind. Somit lautet die Äquivalenzklasse von ϵ:

$[\epsilon] = \{y \in \Sigma^* | N_a(y) = N_b(y)\}$.

$az \in L \Leftrightarrow N_a(z) = N_b(z) - 1$. So verhalten sich genau die Wörter $y \in \Sigma^*$, die ein *a* mehr als *b*'s enthalten. Somit lautet die Äquivalenzklasse von *a*:

$[a] = \{y \in \Sigma^* | N_a(y) = N_b(y) + 1\}$.

Wir können in diesem Stil fortfahren und für jedes $a^i, i > 1$ eine eigene Äquivalenzklasse definieren:

$[a^i] = \{y \in \Sigma^* | N_a(y) = N_b(y) + i\}, i > 1$.

Das sind zwar noch längst nicht alle Äquivalenzklassen, aber wir haben genug gesehen, um folgern zu können, dass es eine unendliche Zahl von Äquivalenzklassen gibt, der Index von *L* also unendlich ist.

Wir betrachten nun ein Beispiel für eine nicht-reguläre Sprache über $\Sigma = \{a, b, c\}$, die die Bedingungen des Pumping Lemma erfüllt. Sie lautet

$$L = \{x \in \Sigma^* \mid x = a^m b^n c^n \text{ mit } m, n \geq 1\} \cup$$
$$\{x \in \Sigma^* \mid x = b^m c^n \text{ mit } m, n \geq 0\}$$

Die Wörter dieser Sprache können also eine von zwei Gestalten annehmen: Sie können mit einer beliebig langen Folge von *a*'s beginnen, auf die eine gleiche Anzahl von *b* und *c* folgen muss. Falls keine *a*'s am Anfang des Worts stehen, besteht das Wort aus Folgen von *b* und *c*, die unterschiedlich lang sein dürfen. In diesem Teil der Sprache ist auch das leere Wort enthalten. Mit Hilfe des Satzes von Myhill-Nerode zeigen wir zunächst, dass *L* nicht-regulär ist. Dazu betrachten wir Wörter der Form ab^i für $i \geq 0$. Wörter $\in L$, die mit *a* beginnen, müssen am Ende gleich lange *a*- und *b*-Strings enthalten:

$ab^i z \in L \Leftrightarrow z = b^{n-i} c^n$. Genau so verhalten sich die Wörter $\in \Sigma^*$ der Form $a^m b^i$:

$[ab^i] = \{y \in \Sigma^* | y = a^m b^i \text{ mit } m \geq 1\}$.

Da wir für jedes *i* eine unterschiedliche Äquivalenzklasse erhalten, ist der Index von *L* unendlich.

Tatsächlich kann man von dieser Sprache zeigen, dass sie, obwohl nicht-regulär, die Bedingungen des Pumping Lemma erfüllt. Dazu wählen wir $N = 1$. Wir betrachten also alle $x \in L$ mit $|x| \geq 1$, mit anderen Worten: Alle Wörter $x \in L$ mit Ausnahme des leeren Worts. Alle diese x müssen sich in der Form $x = uvw$ zerlegen lassen mit $|uv| \leq 1, |v| \geq 1, uv^i w \in L$. Die ersten beiden Bedingungen erzwingen $u = \epsilon$. Dann ist $|uv| = |v| \leq 1$, wenn wir als v das erste Symbol von x wählen. Wir können demnach drei Fälle unterscheiden:

- ✔ $v = a$

  Dann hat $x \in L$ die Form $x = aa^{m-1}b^n c^n$ mit $m \geq 1, n \geq 1$. Offenbar lässt sich das führende a, also der Schleifenbestandteil v, beliebig aufpumpen oder man kann es weglassen, ohne dass das Wort aus L herausfällt.

- ✔ $v = b$

  In diesem Fall sind wir im zweiten Teil der Sprache: $x = bb^{m-1}c^n$ mit $m \geq 1, n \geq 0$. Da die Anzahl der b's und c's nicht übereinstimmen muss, lässt sich auch in diesem Fall das führende Symbol b beliebig oft wiederholen, oder man kann es weglassen.

- ✔ $v = c$

  Auch hier sind wir im zweiten Sprachenteil. Da $x \in L$ und mit c beginnt, kann x nur die Form $x = cc^{n-1}, n \geq 1$ haben. Offenbar lässt sich das führende c beliebig aufpumpen, oder man kann es weglassen.

Dieses zugegebenermaßen etwas konstruierte Beispiel zeigt, dass es nicht-reguläre Sprachen geben kann, die trotzdem das Pumping Lemma erfüllen können. Die Nicht-Regularität rührt wie eh und je daher, dass ein DFA nicht prüfen kann, ob ein Eingabewort gleich viele b und c enthält. Trotzdem erfüllt die Sprache das Pumping Lemma, da beide Sprachteile zu Beginn einen pumpbaren Bestandteil aufweisen.

Nun zum Beweis des Satzes von Myhill-Nerode. Zur Erinnerung: Dieser Satz besagt, dass eine Sprache L genau dann regulär ist, wenn ihr Index, also die Zahl der Äquivalenzklassen der Relation R_L endlich ist. Wir können sogar noch genauer werden:

Index und Minimalautomat einer Sprache

Ist L eine reguläre Sprache, so gibt ihr Index die Anzahl der Zustände des Minimalautomaten für L an.

Der Beweis läuft folgendermaßen ab: Zunächst konstruieren wir aus den Äquivalenzklassen von R_L einen DFA M, der L akzeptiert und der genau so viele Zustände wie Äquivalenzklassen besitzt. Dann zeigen wir, dass M Minimalautomat zu L ist.

Sei $L \subset \Sigma^*$ also irgendeine Sprache. Ist ihr Index endlich, so können wir einen DFA $M = (Z, \Sigma, z_0, \delta, F)$ definieren, indem wir jeder Äquivalenzklasse $[u]$ einen Zustand $z_{[u]}$ zuordnen. Startzustand z_0 ist dabei der Zustand $z_{[\epsilon]}$, Endzustände sind alle Zustände $z_{[u]}$ mit $u \in L$. Die Übergangsfunktion δ definieren wir über

$$\forall u \in \Sigma^* \forall a \in \Sigma : \delta(z_{[u]}, a) = z_{[ua]}$$

Wir müssen noch nachprüfen, dass δ dadurch wohldefiniert ist, d. h., dass der Funktionswert für alle Wörter $w \in [u]$ gleich ist. Betrachten wir also zwei unterschiedliche Wörter $w_1, w_2 \in [u]$. Dann gilt, je nachdem welches Wort wir als Repräsentanten von $[u]$ auswählen, $\delta(z_{[u]}) = z_{[w_1 a]}$ oder $\delta(z_{[u]}) = z_{[w_2 a]}$. Weil aber w_1 und w_2 in derselben Äquivalenzklasse $[u]$ sind, sind sie zueinander äquivalent, und damit sind auch $w_1 a$ und $w_2 a$ äquivalent. Also ist $[w_1 a] = [w_2 a]$ und δ wohldefiniert.

Was bedeutet das für die erweiterte Übergangsfunktion $\hat{\delta}$? Sei $u = a_1 a_2 \dots a_n \in \Sigma^*$. Dann gilt:

$$\begin{aligned}\hat{\delta}(z_0, u) &= \hat{\delta}(z_{[\epsilon]}, a_1 a_2 \dots a_n) \\ &= \delta(\hat{\delta}(z_{[\epsilon]}, a_1 a_2 \dots a_{n-1}), a_n) \\ &= \dots = \delta(\delta(\dots \delta(\delta(z_{[\epsilon]}, a_1), a_2) \dots, a_{n-1}), a_n)\end{aligned}$$

Jetzt können wir von innen nach außen die Funktionswerte für δ berechnen, beginnend bei $\delta(z_{[\epsilon]}, a_1) = z_{[\epsilon a_1]} = z_{[a_1]}$. Am Schluss erhalten wir den Zustand $z_{[a_1 a_2 \dots a_n]} = z_{[u]}$ als Ergebnis. Damit haben wir ein wichtiges Ergebnis erhalten:

$$\forall u \in \Sigma^* : \hat{\delta}(z_0, u) = z_{[u]}$$

Jetzt gilt:

$$x \in L \Leftrightarrow z_{[x]} \in F \Leftrightarrow \hat{\delta}(z_0, x) \in F \Leftrightarrow x \in L(M).$$

Unser Äquivalenzklassen-DFA M akzeptiert also tatsächlich L, und L ist damit regulär, wenn ihr Index endlich ist.

Jetzt sei umgekehrt L eine reguläre Sprache. Wir zeigen, dass der Äquivalenzklassen-Automat M von L sogar der Minimalautomat zu L ist, denn zum einen enthält er keine nicht erreichbaren Zustände (anderenfalls müsste es eine Äquivalenzklasse geben, in der kein Wort enthalten ist). Zum anderen enthält er keine äquivalenten Zustände, die man zusammenlegen könnte:

Angenommen, $z_{[u]}$ und $z_{[v]}$ sind äquivalente Zustände. Das bedeutet, erklärte man $z_{[u]}$ bzw. $z_{[v]}$ zum Startzustand von M, würde M die gleiche Sprache akzeptieren: $L(M, z_{[u]}) = L(M, z_{[v]})$. Anders gesagt: Startet man bei $z_{[u]}$ bzw. $z_{[v]}$ und gibt man ein Wort w in den Automaten ein, so ist das Resultat in beiden Fällen gleich: $\hat{\delta}(z_{[u]}, w) \in F \Leftrightarrow \hat{\delta}(z_{[v]}, w) \in F$. Wegen $\hat{\delta}(z_0, u) = z_{[u]}$ können wir weiter folgern:

$$\forall w \in \Sigma^* : \hat{\delta}(\hat{\delta}(z_0, u), w) \in F \Leftrightarrow \hat{\delta}(\hat{\delta}(z_0, v), w) \in F, \text{ also}$$

$$\forall w \in \Sigma^* : \hat{\delta}(z_0, uw) \in F \Leftrightarrow \hat{\delta}(z_0, vw) \in F, \text{ also}$$

$$\forall w \in \Sigma^* : uw \in F \Leftrightarrow vw \in F.$$

Das bedeutet, u und v sind äquivalente Wörter. Also sind ihre Äquivalenzklassen gleich: $[u] = [v]$ und damit auch $z_{[u]} = z_{[v]}$.

M ist also der Minimalautomat von L, wobei die Zustände von M den Äquivalenzklassen von R_L entsprechen. Der Index von L ist damit endlich und entspricht der Zahl der Zustände des Minimalautomaten.

Lassen Sie uns die doch recht abstrakte Konstruktion des Minimalautomaten mit Hilfe der Äquivalenzklassen von L an einem Beispiel nachvollziehen.

Wir betrachten die Sprache der binären Strings, die eine ungerade Anzahl von Einsen enthalten: $\Sigma = \{0, 1\}, L = \{w \in \Sigma^* | N_1(w)$ ungerade.$\}$ Die Äquivalenzklassen von R_L sind: $[\epsilon] = \{x \in \Sigma^* | N_1(x)$ gerade$\}$ und $[1] = \{x \in \Sigma^* | N_1(x)$ ungerade $\}$. Daraus können wir einen zu L passenden Automaten $M = (Z, \Sigma, z_0, \delta, F)$ konstruieren mit $Z = \{z_{[\epsilon]}, z_{[1]}\}, z_0 = z_{[\epsilon]}$ und $F = \{z_{[1]}\}$.

Die Übergangsfunktion δ wird festgelegt durch die Bedingung $\hat{\delta}(z_0, u) = z_{[u]}$:

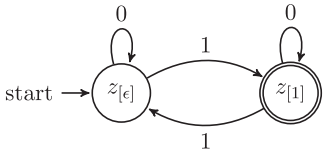
$\delta(z_{[\epsilon]}, 0) = \hat{\delta}(z_0, 0) = z_{[0]} = z_{[\epsilon]}$

$\delta(z_{[\epsilon]}, 1) = \hat{\delta}(z_0, 1) = z_{[1]}$

$\delta(z_{[1]}, 0) = \delta(\hat{\delta}(z_0, 1), 0) = \hat{\delta}(z_0, 10) = z_{[10]} = z_{[1]}$

$\delta(z_{[1]}, 1) = \delta(\hat{\delta}(z_0, 1), 1) = \hat{\delta}(z_0, 11) = z_{[11]} = z_{[\epsilon]}$

Der dazugehörige Übergangsgraph sieht so aus:



DFAs mit Ausgabe (Moore- und Mealy-Automaten)

Die DFAs, die wir bis jetzt betrachtet haben, akzeptierten entweder einen Eingabestring w, oder sie akzeptierten ihn nicht. Deshalb werden diese DFAs auch als **Akzeptoren** bezeichnet. Wir schauen uns nun einen anderen Typ von DFA an, der zu jedem Eingabezeichen ein Ausgabezeichen liefert und so einen Eingabestring nach und nach gewissermaßen in einen Ausgabestring übersetzt. Solche Automaten nennt man **Transduktoren**.

Transduktoren sind eine einfache Erweiterung von Akzeptoren. Der Mechanismus der Zustandswechsel aufgrund von Eingaben bleibt der gleiche. Ein wichtiger Unterschied zu den Akzeptoren besteht jedoch darin, dass Transduktoren keine ausgewiesenen Endzustände

besitzen. Solange wir das richtige Eingabealphabet Σ benutzen, ist also jede Eingabe gültig und wird vom Automaten nach einem vorgegebenen Regelwerk in eine Ausgabe übersetzt. Die Ausgabe wird durch eine Ausgabefunktion ζ geregelt, die Zeichen aus einem **Ausgabealphabet** Y ausgibt. Hängt die Ausgabefunktion sowohl vom Eingabezeichen als auch vom Zustand des Automaten ab, gilt also $\zeta : Z \times \Sigma \to Y$, nennt man den Automaten einen **Mealy-Automaten**.

Hängt die Ausgabefunktion dagegen nur vom Zustand des Automaten ab, gilt also $\zeta\colon Z \to Y$, nennt man den Automaten einen **Moore-Automaten**. Manche Aufzüge verfügen über eine Anzeige, die das Stockwerk angibt, in dem sich der Aufzug aktuell befindet. Ein solcher Aufzug stellt ein sehr einfaches Beispiel für einen Moore-Automaten dar: Die Ausgabe entspricht der Nummer des aktuellen Zustands des Automaten und hängt damit nur vom aktuellen Zustand ab.

Jetzt schauen wir uns Beispiele für Mealy-Automaten an und untersuchen gleichzeitig, wie sich die Ausgabefunktion ζ im Übergangsdiagramm des Automaten widerspiegelt:

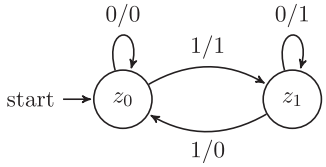
Ein *Flip-Flop* ist ein grundlegendes Speicherelement aus der Schaltungstechnik, das ein Bit speichern kann. Es besitzt zwei stabile Zustände, die mit *Set* (gesetzt) und *Reset* (zurückgesetzt) bezeichnet werden. Legt man eine Eingangsspannung an das Element an, wechselt es den Zustand. Modelliert man das Anlegen von Spannung mit dem Eingabesymbol 1 und keine Spannung mit 0, so ergibt sich daraus die folgende Übergangsfunktion über dem Alphabet $\Sigma = \{0, 1\}$:

δ	0	1
z_0	z_0	z_1
z_1	z_1	z_0

wobei z_0 den Reset-Zustand und z_1 den Set-Zustand symbolisiert. Das Auslesen eines Bits aus dem Speicherelement erfolgt nach dem Anlegen einer Spannung bzw. Nicht-Spannung. Es entspricht dann dem Index des neuen Zustands. Das Ausgabealphabet ist also $Y = \{0, 1\}$, und die Ausgabefunktion $\zeta\colon Z \times \Sigma \to Y$ sieht folgendermaßen aus:

ζ	0	1
z_0	0	1
z_1	1	0

Die Information über die Werte der Ausgabefunktion wird in das Übergangsdiagramm aufgenommen, indem man über jede Kante $\delta(z_i, a) = z_j$ neben $a \in \Sigma$ noch zusätzlich das Ausgabesymbol $y = \zeta(z_i, a) \in Y$ schreibt:



Mit einem *sequenziellen Addierer* kann man zwei gleich lange Binärzahlen x und y addieren, indem man jeweils Paare gleichwertiger Bits von x und y, beginnend bei den Bits mit der niedrigsten Wertigkeit, in den Automaten eingibt. Das Eingabealphabet ist somit $\Sigma = \{0,1\}^2 = \{(0,0),(0,1),(1,0),(1,1)\}$. Bei der binären Addition gilt bekanntlich

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0 +$ Übertragsbit

Solange kein Übertrag auftritt, können wir im Startzustand z_0 bleiben und geben mittels ζ jeweils das Ergebnisbit auf der rechten Seite der Gleichungen aus. Im Falle eines Übertrags wechseln wir in den Zustand z_1. Im neuen Zustand gelten jetzt andere Additionsregeln, da wir das Übertragsbit berücksichtigen müssen:

$0 + 0 = 1$

$0 + 1 = 0 +$ Übertragsbit

$1 + 0 = 0 +$ Übertragsbit

$1 + 1 = 1 +$ Übertragsbit

Solange ein Übertragsbit benötigt wird, bleiben wir in z_1. Nur im Fall der Eingabe $(0,0)$ wechseln wir wieder nach z_0. Insgesamt ergibt sich der folgende Mealy-Automat:

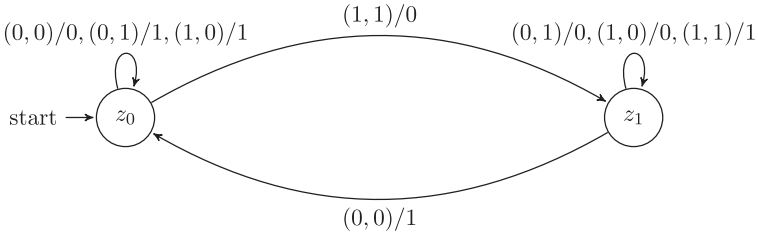


Abbildung 1.6: Ein sequenzieller Addierer als Mealy-Automat

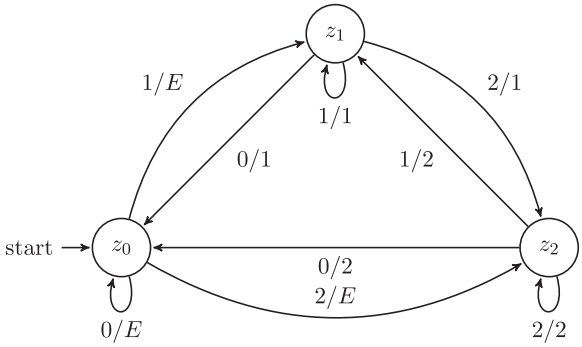
Es ist klar, dass sich jeder Moore-Automat als ein spezieller Mealy-Automat auffasssen lässt. Die beiden Automaten-Modelle sind aber nicht gleichwertig: Während Moore-Automaten für die Ausgabe nur die Information über den aktuellen Zustand nutzen, verfügen Mealy-Automaten zusätzlich über die Information über das nächste Eingabezeichen, können also gewissermaßen in die Zukunft schauen.

Da jeder Moore-Automat im Prinzip auch einen Mealy-Automat darstellt, ist beim Betrachten von Übergangsdiagrammen wie in den obigen drei Beispielen nicht unmittelbar klar, um welchen Automatentyp es sich handelt, es sei denn, man benutzt für Moore-Automaten eine andere Form der Darstellung für die Ausgabe:

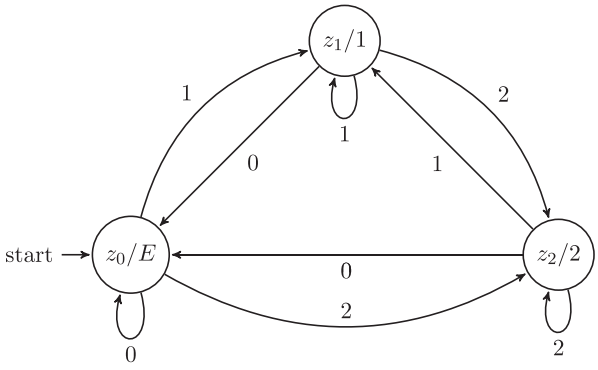
Betrachten wir als Beispiel für einen Moore-Automaten einen Aufzug mit Stockwerksanzeige. Wir nehmen an, es gibt ein Erdgeschoss und zwei Obergeschosse. Dann ist $Y = \{E, 1, 2\}, Z = \{z_0, z_1, z_2\}$, und die Output-Werte der Ausgabefunktion $\zeta : Z \to Y$ lauten $\zeta(z_0) = E, \zeta(z_1) = 1, \zeta(z_2) = 2$. Genauso gut könnte man ζ aber auch in dieser Form angeben:

ζ	E	1	2
z_0	E	E	E
z_1	1	1	1
z_2	2	2	2

Hier scheint es, als würde $\zeta : Z \times \Sigma \to Y$ gelten. Aus der Tabelle ergibt sich der folgende Übergangsgraph:



Wieder hat man auf den ersten Blick den Eindruck, es handle sich um einen Moore-Automaten. Besser ist es, man bindet im Falle eines Mealy-Automaten die Ausgabe direkt an die Zustandsknoten:



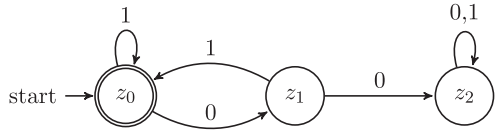
Leider sind die Definition und die Darstellungsform für Moore- und Mealy-Automaten in der Literatur nicht ganz eindeutig. Sie sollten also immer ganz genau hinschauen, wie die Ausgabefunktion ζ arbeitet.

Aufgaben zu DFAs

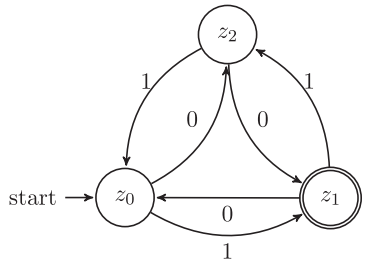
Als kleine Hilfe, so zur Sicherheit für Sie, stelle ich Ihnen einige der Lösungen unter `http://wiley-vch.de/ISBN978-3-527-71431-5` zur Verfügung, aber nicht alle. Einige Lösungen müssen Sie selbst ausknobeln, so bleiben sie auch besser im Gedächtnis haften.

1. Genau betrachtet, bildet der Wolf-Ziege-Kohlkopf-Automat, so wie wir ihn im Beispiel gezeichnet haben, **keinen** DFA. Erklären Sie, warum nicht.
2. Ein Getränkeautomat akzeptiert 1-Euro- und 50-Cent-Münzen. Ein Getränk soll 1,50 Euro kosten. Modellieren Sie den Getränkeautomaten als DFA.
3. Welche Sprachen akzeptieren die folgenden DFAs über $\Sigma = \{0, 1\}$?

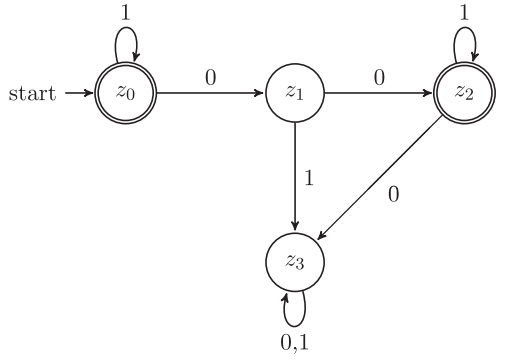
a.



b.



c.



4. Geben Sie jeweils einen DFA an, der die folgenden Sprachen über $\Sigma = \{0, 1\}$ akzeptiert:
   a. $L = \{01, 0011, 000111\}$
   b. $L = \{w \in \Sigma^* | w \text{ enthält 101 als Substring.}\}$

c. $L = \{w \in \Sigma^* | w$ beginnt und endet mit $1.\}$

d. $L = \{w \in \Sigma^* |$ jeder 1 in w geht mindestens eine 0 voraus und es folgt mindestens eine 0 nach.$\}$

5. Σ sei ein Alphabet. Zeigen Sie, dass Σ^* mit der Operation Konkatenation ein Monoid bildet.

6. $M = (Z, \Sigma, \delta, z_0, F)$ sei ein DFA. Zeigen Sie, dass für die erweiterte Übergangsfunktion $\hat{\delta}$ gilt:

$\forall z \in Z \forall a \in \Sigma : \hat{\delta}(z, a) = \delta(z, a)$

7. Zeigen Sie:

Wenn ein DFA M mit n Zuständen ein Wort w mit $|w| \geq n$ akzeptiert, dann enthält $L(M)$ unendlich viele Wörter.

8. Zeigen Sie mit Hilfe des Pumping Lemma, dass die Sprache

$L = \{w \in \Sigma^* | w$ enthält mehr Nullen als Einsen$\}$

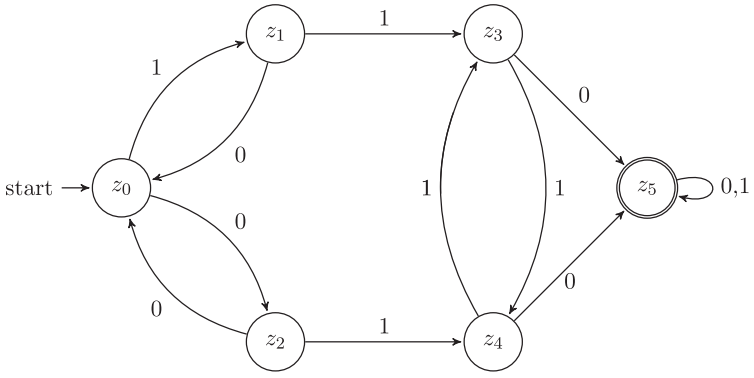
über dem Alphabet $\Sigma = \{0, 1\}$ nicht-regulär ist.

9. Zeigen Sie mit Hilfe des Satzes von Myhill-Nerode, dass die Sprache

$L = \{w \in \Sigma^* | w = 1^{3n}, n \geq 0\}$

über dem Alphabet $\Sigma = \{1\}$ regulär ist, und geben Sie einen DFA M an mit $L(M) = L$.

10. Finden Sie den Minimalautomaten zu diesem DFA :



11. Finden Sie einen DFA über $\Sigma = \{a, b\}$, der die Sprache

$L = \{w \in \Sigma^* | w$ beginnt und endet mit dem gleichen Symbol und $|w| \geq 2\}$

akzeptiert.