Automaten und Grammatiken

Fritz Hasselhorn

20. Dezember 2022

Inhaltsverzeichnis

1 Endliche Automaten

In diesem Kapitel lernen Sie

- wie ein endlicher Automat definiert ist und wie er funktioniert,
- wie man deterministische und nichtdeterministische Automaten voneinander abgrenzt,
- aus welchen Bestandteilen sich endliche Automaten konstruieren lassen,
- wie man komplexe endliche Automaten aus einfachen zusammensetzt,

1.1 Allgemeines

Ein endlicher Automat ist ein mathematisches Modell eines Systems mit diskreten Ein- und Ausgaben. Diskret bedeutet, dass zu jedem Zeittakt nur eine Eingabe verarbeitet wird und eine Ausgabe ausgegeben wird. Es gibt endlich viele verschiedene Eingaben und endlich viele verschiedene Ausgaben. Das System befindet sich jeweils in einem von endlich vielen Zuständen. In den Zuständen werden die Informationen gespeichert, die sich aus den bisherigen Eingaben ergeben und die für die weiteren Reaktionen des Systems notwendig sind. So speichert z.B. ein Fahrstuhl das aktuelle Stockwerk und die Fahrtrichtung (aufwärts oder abwärts).

Die Theorie der endlichen Automaten beschreibt, wie man solche Systeme entwirft. In der Informatik gibt es viele Beispiele für Systeme mit endlicher endlicher Zustandsmenge. So werden elektronische Schaltnetze so entwickelt, dass man sie als Automaten beschreiben kann. Auch elektronische Speicher (Flipflops) lassen sich als Automaten beschreiben.

Die Einsatzmöglichkeit von endlichen Automaten beschränkt sich aber keineswegs auf Hardwaresysteme. Programmiersprachen verfügen über einen Compiler, der prüft, ob der Programmtext die Syntaxregeln der jeweiligen Computersprache einhält, und gegebenenfalls Fehlermeldungen ausgibt, z.B. „Semikolon erwartet“. Der erste Compiler wurde 1952 von der Mathematikerin Grace Hooper entwickelt.

Bei der Überprüfung des Programmtextes muss sich der Compiler eine endliche Menge von Informationen merken, z.B. dass mit dem Schlüsselwort „VAR“ eine Deklaration von Variablen eingeleitet wurde.

Man könnte den Computer selbst als ein System mit einer endlichen Zustandsmenge beschreiben, weil der konkrete Speicher jedes Computers endlich ist. Die Automatentheorie stellt aber andere Modelle mit prinzipiell unendlichem Speicher zur Verfügung, die die Leistungsfähigkeit eines Computers besser beschreiben. Ein endlicher Automat kann aufgefasst werden als ein Computer mit einem einzigen Speicherplatz, wobei der Speicher den aktuellen Zustand enthält.

Wie leistungsfähig das Konzept der endlichen Automaten ist, soll an einem bekannten Beispiel aus der Logik betrachtet werden: Ein Mann steht mit einem Wolf, einer Ziege und einem Kohlkopf am linken Ufer eines Flusses. Er will mit einem kleinen Boot übersetzen. Das Boot fasst außer ihm aber höchstens ein weiteres Objekt, also entweder den Wolf, die Ziege oder den Kohlkopf. Er darf den Wolf mit der Ziege nicht allein an einem Ufer zurücklassen, weil sonst der Wolf die Ziege frisst. Bleibt die Ziege allein mit dem Kohlkopf an einem Ufer, so frisst sie ihn ebenfalls auf.

In diesem Beispiel gibt es 16 verschiedene Zustände mit den Elementen „Mann (M)“, „Wolf (W)“, „Ziege (Z)“ und „Kohlkopf (K)“. Die Zustände werden in der Weise notiert, dass die Objekte am linken Ufer links vom Bindestrich und die Objekte am rechten Ufer rechts vom Bindestrich stehen. So bedeutet„WK-MZ“: Wolf und Kohl am linken Ufer, Mann und Ziege am rechten Ufer. Die

„Eingaben“ sind die durchgeführten Aktionen. Der Mann kann allein den Fluss überqueren (m), mit dem Wolf (w), mit der Ziege (z) oder mit dem Kohlkopf (k). Der Startzustand ist „MWKZ−∅ “, wobei ∅ für die leere Menge steht, d.h. am rechten Ufer befindet sich kein Objekt. Nach jedem Übersetzen wird der neue Zustand notiert. Der gewünschte Endzustand ist „∅−MWKZ“.

Zum Zeichnen und Testen von Automaten setzen wir das Programm AutoEdit ein. Das Programm ermöglicht nicht nur die Konstruktion von deterministischen endlichen Automaten (DEA) und Mealy-Automaten, die wir in diesem Kapitel behandeln, sondern auch aller weiterer Automaten, wie nichtdeterministische endliche Automaten (NEA) und Kellerautomaten, die in der Kursstufe behandelt werden. Das Programm AutoEdit ist Teil des Programmpaketes AtoCC, das man kostenlos nutzen darf. Man findet es, wenn man im Internet nach „AtoCC“ sucht.

Als Alternative haben die Programmierer von AtoCC eine Webseite programmiert, die unter **www.flaci.com** erreichbar ist. Sie bietet die Möglichkeit, Automaten und Grammatiken zu erstellen, zu speichern und zu testen. Allerdings sind Kommas unter flaci.com als Trennzeichen reserviert.

Aufgabe 1.1 *Mann-Wolf-Ziege-Kohl*

1. *Notiere die 16 verschiedenen Zustände. Markiere diejenigen sechs Zustände, die vermieden werden müssen.*

2. *Zeichne das Zustandsdiagramm eines endlichen Automaten aus den verbleibenden 10 Zuständen und den zugehörigen Übergängen, so dass sich die Wege vom Startzustand bis zum Endzustand ablesen lassen. Zeichne dabei nur die Übergänge ein, die nicht in einen ausgeschlossenen Zustand führen.*

1.2 DEA

1.2.1 Definition

Ein deterministischer endlicher Automat (DEA) besteht aus einer endlichen Menge von Eingaben (dem Eingabealphabet) und einer endlichen Zustandsmenge. Ein Zustand ist der Startzustand. Einer oder mehrere Zustände sind akzeptierende Zustände. Außerdem gehört zum Automaten eine Überführungsfunktion, die zu jedem Zustand und zu jedem Eingabezeichen den entsprechenden Folgezustand angibt. Formal lässt sich ein deterministischer endlicher Automat als Quintupel definieren:

Definition:
Ein **deterministischer endlicher Automat (DEA)** besteht aus

1. einer endlichen Menge von Zuständen Q,

2. einem Eingabealphabet Σ,

3. einer eindeutigen Überführungsfunktion δ, die zu einem Zustand und einem Eingabezeichen genau einen Folgezustand berechnet, wobei die Überführungsfunktion nicht vollständig sein muss,

4. einem Startzustand (hier q_0),

5. einer endlichen Menge von akzeptierenden Zuständen E (Der häufig verwendete Begriff „Endzustände“ ist missverständlich, weil der Automat je nach Eingabewort in jedem Zustand enden kann.)

Ein DEA dient zur Überprüfung einer Folge von Eingabezeichen, die gemeinsam ein Wort bilden. Der DEA prüft, ob das jeweils gegebene Wort dazu führt, dass der DEA vom Startzustand in einen akzeptierenden Zustand (Endzustand) übergeht oder in einen anderen Zustand. Alle Worte, also alle Folgen von Eingabezeichen, die in einem akzeptierenden Zustand enden, gehören zur Sprache des DEA. Der DEA beginnt im Startzustand. Er geht dann das Wort Zeichen für Zeichen durch und berechnet mit Hilfe der Überführungsfunktion jeweils aus aktuellem Zustand und aktuellem Eingabezeichen den Folgezustand.

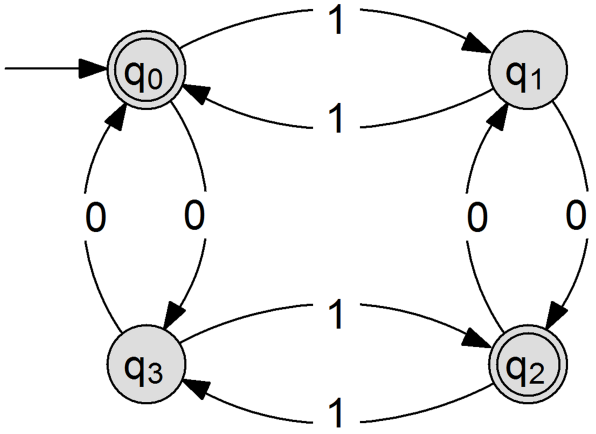

Abbildung 1.1: Zustandsdiagramm eines DEA

Ein deterministischer endlicher Automat kann mit einem Zustandsdiagramm beschrieben werden. Ein Zustandsdiagramm ist ein gerichteter Graph, bestehend aus Knoten und Kanten mit einer Richtung, die durch die Pfeilspitzen ausgedrückt wird. Die Knoten (Kreise) entsprechen den Zuständen. Der Startzustand ist mit einem zusätzlichen Pfeil auf den Kreis und dem Wort „Start“ gekennzeichnet (hier q_0). Die akzeptierenden Zustände sind durch einen Doppelkreis gekennzeichnet (hier q_0 und q_2). Die Eingaben stehen an den Kanten (Pfeilen). Der Pfeil von q_0 nach q_1 ist mit „1“ beschriftet. Das bedeutet: „Befindet sich der Automat im Zustand q_0, dann geht er durch Eingabe von 1 in den Zustand q_1“.

Die Überführungsfunktion lässt sich auch als Tabelle schreiben:

δ	0	1
q_0	q_3	q_1
q_1	q_2	q_0
q_2	q_1	q_3
q_3	q_0	q_2

Damit lautet die Definition des unter 2.1. abgebildeten Automaten:

- Zustandsmenge Q = $\{q_0, q_1, q_2, q_3\}$
- Eingabealphabet $\Sigma = \{0, 1\}$
- Überführungsfunktion δ (wie oben abgebildet)
- Startzustand q_0
- Menge der akzeptierenden Zustände E = $\{q_0, q_2\}$

1.2.2 Funktionsweise

Bildlich kann man sich einen endlichen Automaten vorstellen als Blackbox, die sich in einem Zustand aus der Zustandsmenge S befindet und eine Folge von Symbolen aus dem Eingabealphabet E liest, die - wie in der Abbildung gezeigt - auf einem Band stehen. Dabei kann sich das Eingabeband nur in eine Richtung bewegen.

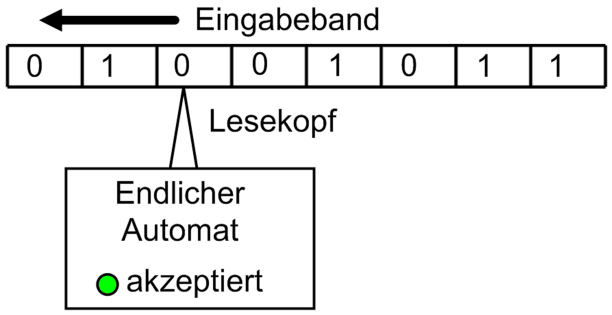

Abbildung 1.2: Aufbau eines DEA

Der DEA beginnt im Startzustand. In einem Schritt geht der endliche Automat, der sich im Zustand s_i befindet und das Eingabezeichen e_k liest, in den neuen Zustand s_j, den die Überführungsfunktion aus altem Zustand und Eingabezeichen berechnet, und bewegt seinen Lesekopf um ein Eingabezeichen nach rechts. Ist das Wort auf dem Eingabeband abgearbeitet und der Automat befindet sich in einem akzeptierenden Zustand, so gehört das Wort zu der Sprache, die der endliche Automat akzeptiert. Alle Worte, also alle Folgen von Eingabezeichen, die in einem akzeptierenden Zustand enden, gehören zur Sprache des DEA.

Aufgabe 1.2 *Eingabeworte für die verschiedenen Zustände*

Gegeben ist der Automat in Abbildung 1.1. Gib zu jedem Zustand ein Eingabewort an, so dass der Automat in diesem Zustand endet. Markiere dabei diejenigen Eingabeworte, die vom Automaten akzeptiert werden.

1.2.3 Determinismus

Deterministisch bedeutet, dass das Verhalten des Automaten in jeder Situation eindeutig festgelegt ist. Von jedem Zustand aus darf es nur eine Kante mit jedem Eingabezeichen geben. Man sieht am einfachsten, ob es sich um einen deterministischen endlichen Automaten handelt, in dem man die Tabelle der Überführungsfunktion betrachtet (unter Auto-Edit erreichbar über „Exportieren“).

δ	0	1
q_0	q_3	q_1
q_1	q_2	q_0
q_2	q_1, q_2	q_3
q_3	q_0	q_2

Man sieht hier, dass im Zustand q_2 zwei Kanten mit „0“abgehen, nämlich nach q_1 und q_2. Damit handelt es sich nicht mehr um einen deterministischen, sondern um einen nichtdeterministischen endlichen Automaten (NEA). Dafür reicht schon ein einziger Verstoß aus. Mit nichtdeterministischen endlichen Automaten und deren Überführung in deterministische werden wir uns in der Kursstufe befassen. Sie sind nicht nur eine Spielerei, sondern bestens geeignet, die Konstruktion von endlichen Automaten zu unterstützen.

Ist die Tabelle der Übergangsfunktion nicht gegeben, dann muss man zur Überprüfung des Determinismus jeden Zustand daraufhin testen, ob hier zwei Kanten mit dem gleichen Eingabezeichen abgehen.

1.2.4 Vollständigkeit

Ein endlicher Automat ist vollständig, wenn in jedem Zustand für jedes Eingabezeichen eine Kante vorhanden ist. Auch hier überprüft man die Eigenschaft am besten mit der Tabelle der Überführungsfunktion:

δ	0	1
q_0	q_1	
q_1	q_1	q_1

Man sieht, dass keine Kante von q_0 mit dem Eingabezeichen „1“ vorliegt.

Häufig verlangen wir von deterministischen endlichen Automaten auch die Vollständigkeit. Teilweise wird in Aufgabenstellungen aber ausdrücklich auf die Darstellung der Fehlerzustände und der Übergänge in die Fehlerzustände verzichtet, wie z.B. in der Einleitung zu diesem Kapitel in der Mann-Wolf-Ziege-Kohl-Aufgabe.

1.2.5 Konstruktion von DEAs

Deterministische endliche Automaten lassen sich aus wenigen Grundbausteinen zusammensetzen. Zu diesen Bausteinen gehören Wiederholung, Verzweigung und Zählen.

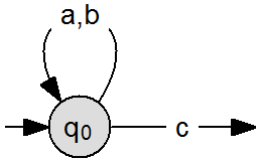

Abbildung 1.3: 0 bis n Wiederholungen

In dem Ausschnitt eines DEAs in Abbildung 2.3 können a und b beliebig oft wiederholt werden. Es ist aber nicht sichergestellt, dass a und b überhaupt vorkommen, weil man auch gleich den Übergang mit c nehmen kann. Wenn man sicherstellen will, dass mindestens eine Wiederholung erfolgt, muss man einen Übergang davorsetzen:

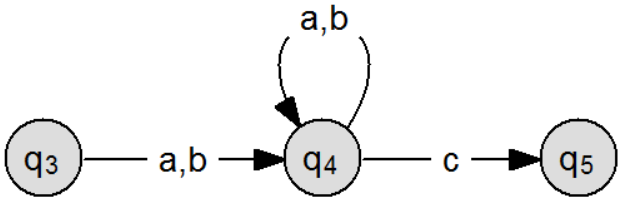

Abbildung 1.4: 1 bis n Wiederholungen

Beide Arten von Wiederholungen verhindern übrigens das Zählen von a und b. Die Konstruktion erlaubt zwar Wiederholungen von a und b, aber sie zählt nicht mit, wie viele Wiederholungen bereits absolviert wurden.

Ein weiterer Grundbaustein ist die Verzweigung:

In Abbildung 2.5 kann entweder mit a bei q_3 fortgesetzt werden oder mit b oder c bei q_2.

Die Verzweigung wird manchmal verbunden mit einem absorbierenden Fehlerstand, auch

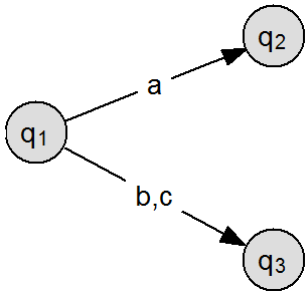

Abbildung 1.5: Verzweigung

Falle (englisch *trap*) genannt. Der Fehlerzustand q_1 heißt absorbierend, weil man ihn nicht mehr verlassen kann. Im Beispiel sind Worte gesucht, die an der ersten Stelle ein a haben. Tritt dort ein b oder c auf, dann gehört das Wort nicht zur Sprache des DEA. Dieser Fehler kann auch nicht mehr korrigiert werden. Auch wenn später ein a auftritt, steht es nicht an der ersten Stelle.

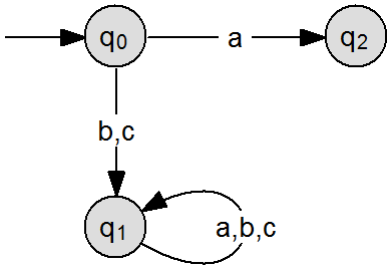

Abbildung 1.6: absorbierender Fehlerzustand

Ein endlicher Automat kann nur zählen, indem er in den nächsten Zustand geht. Da ein endlicher Automat auch nur über endliche viele Zustände verfügt, kann er nicht beliebig weit zählen. Diese Eigenschaft wird in der Kursstufe eine Rolle spielen, wenn wir uns genauer mit den Grenzen endlicher Automaten befassen.

Beim Zählen muss man zwei Arten unterscheiden: Beim fortlaufenden Zählen zählt der Automat 1, 2, usw., er geht also bei jedem Zählschritt in den nächsten Zustand. Daneben gibt es ein wiederholtes Zählen. Z.B. prüft ein Automat, ob die Zahl der Schritte durch 4 teilbar ist. Dann zählt er 1-2-3-4-1-2-3-4-1 usw. Es entsteht ein Zyklus von Zuständen, der immer wieder durchlaufen wird.

Generell lässt sich sagen: beim wiederholten Zählen muss man einen Zyklus von Zuständen bilden, der wiederholt durchlaufen wird. Die Anzahl der Zustände richtet sich danach, wie

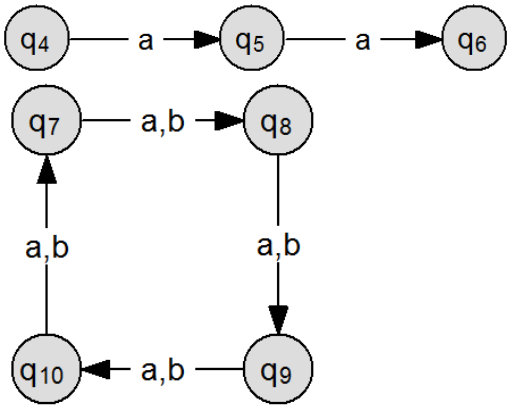

Abbildung 1.7: Fortlaufendes und wiederholtes Zählen

weit gezählt werden soll. Bei der Unterscheidung von gerade und ungerade reichen zwei Zustände. Bei der Frage nach der Teilbarkeit durch Drei reichen drei Zustände. Bei der Teilbarkeit durch Vier reichen vier Zustände, wie in Abbildung 2.7 dargestellt. Auch hier noch einmal der Hinweis: Schleifen von einem Zustand auf sich selbst sind nicht verträglich mit Zählen, weil man dann keine Kontrolle mehr über die Anzahl der Eingabezeichen hat.

1.3 DEA-Grundtypen

Wir betrachten in diesem Kapitel endliche Automaten mit dem Eingabealphabet E={a, b, c}. AutoEdit nummeriert die einzelnen Zustände einfach durch, beginnend mit q_0. Zur Konstruktion eines endlichen Automaten ist es hilfreich, wenn man die Funktion der einzelnen Zustände stichwortartig beschreibt.

Zunächst wollen wir einen Automaten konstruieren, der alle Worte akzeptiert, die mit "ab" beginnen. Der erste Buchstabe soll ein a sein, der zweite ein b. Wir benötigen also außer dem Startzustand q_0 mindestens zwei weitere Zustände: q_1 (1. Buchstabe a) und q_2 (2. Buchstabe b). Von q_0 aus gelangt man mit der Eingabe a in den Zustand q_1 (1. Buchstabe a) und von dort mit b in den Zustand q_2 (2. Buchstabe b). Dabei ist q_2 ein akzeptierender Zustand, weil dort bereits beide Bedingungen erfüllt sind. Was passiert aber, wenn wir im Startzustand b oder c erhalten? Wir können nicht im Startzustand bleiben, weil sonst würde

ein Wort wie bab akzeptiert, obwohl dort weder der 1. Buchstabe ein a ist noch der 2. ein b. Wir können verallgemeinern: Alle Worte, die mit b oder c beginnen, gehören nicht zur Sprache das gesuchten Automaten. Dieser Fehler kann auch nicht mehr nachträglich dadurch korrigiert werden, dass irgendwann später ein ab erscheint. Wir gelangen also durch Eingabe von b oder c im Startzustand in einen Zustand q_3, der eine Falle ist (englisch *Trap*, weil man ihn nicht mehr nachträglich korrigieren kann. Das gleiche gilt im Zustand q_1 für die Eingabe a oder c. Nur mit b kommt man in den Zustand q_2 und erfüllt damit die zweite Bedingung. Die Eingabe von a und c führt in den Fehlerzustand q_3.

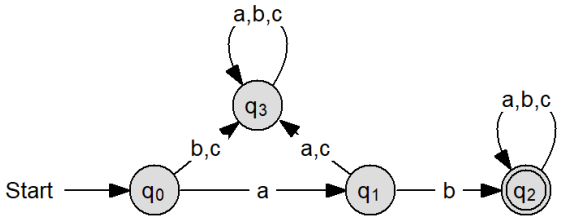

Abbildung 1.8: beginnt mit „ab“

Nun fehlt noch das Verhalten in den Zuständen q_2 und q_3. Wenn wir im Zustand q_2 sind, dann war der erste Buchstabe ein a und der zweite Buchstabe ein b. Unabhängig davon, was jetzt noch folgt, gehört das Wort zur Sprache der Worte, die mit ab beginnen. Mit der Eingabe von a, b oder c bleiben wir im akzeptierenden Zustand q_2. Ähnlich verhält es sich mit dem Zustand q_3. Es handelt sich um eine Falle. Ein Fehler beim 1. oder 2. Buchstaben kann nicht mehr korrigiert werden. Also bleiben wir mit der Eingabe von a, b oder c im Fehlerzustand q_3.

Als nächstes entwickeln wir den Automaten zur Sprache aller Worte, die auf „ab“ enden. Wir benötigen außer dem Startzustand q_0 zwei weitere Zustände q_1 (zuletzt a) und q_2 (zuletzt b, davor a). Der Zustand q_2 ist der akzeptierende Zustand. Mit a kommen wir von q_0 nach q_1, mit b von q_1 nach q_2. Eine Falle wie im vorigen Beispiel kann hier nicht auftreten, weil ja immer noch „ab“ am Ende des Wortes kommen können, egal wie viele Buchstaben davor stehen. Von q_0 aus bleiben wir mit b oder c in q_0, weil unser Muster mit a beginnt. Wenn wir in q_1 ein a bekommen, gilt ja „zuletzt ab“. Wir bleiben also in q_1. Mit b kommen wir nach q_2 und mit c müssen wir zurück nach q_0, weil unser Muster unterbrochen wurde. In q_3 kommen wir mit a zurück nach q_1 (zuletzt a). Mit b und c wird das Muster unterbrochen und wir müssen zurück nach q_0.

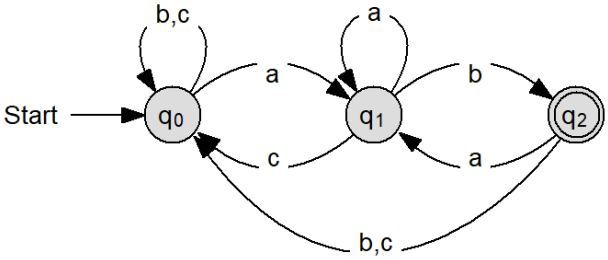

Abbildung 1.9: endet auf „ab“

Mit einem a kommen wir aus allen Zuständen nach q_1 (zuletzt a). Von q_1 aus kommen wir mit b nach q_2. Alle anderen Kanten führen zurück in den Startzustand.

Der dritte Automat soll alle Worte akzeptieren, die mindestens ein „ab“ enthalten. Außer dem Startzustand q_0 gibt es einen Zustand q_1 (zuletzt a) und einen akzeptierenden Zustand q_2 (mindestens ein ab). Aus dem Startzustand gelangen wir mit einem a in den Zustand q_1. Mit einem b oder c bleiben wir in q_0, weil unser Muster ja mit a beginnt. In q_1 bleiben wir mit einem a in q_1, weil der letzte Buchstabe ein a ist. Mit einem b gelangen wir in den Zustand q_2. Mit einem c gelangen wir zurück nach q_0, weil das Muster neu begonnen werden muss. Sind wir in q_2 angelangt, dann enthält unser Wort bereits ein „ab“ und erfüllt die geforderte Bedingung. Was danach kommt, spielt keine Rolle. Wir können also drei Kanten von q_2 nach q_2 zeichnen mit a, b und c.

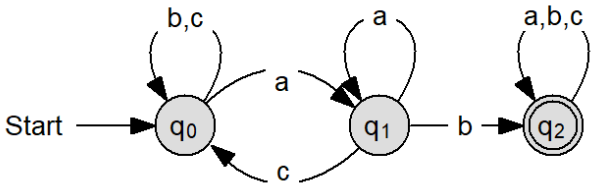

Abbildung 1.10: mindestens ein „ab“

Die beiden Sprachen „enthält mindestens ein ab“ und „enthält kein ab“ sind komplementär zueinander. Das bedeutet, jedes Wort aus a, b und c gehört entweder zu der einen Sprache oder zu der anderen. Alle Worte, die unser dritter Automat (enthält mindestens ein ab) akzeptiert, darf der Automat zu „enthält kein ab“ nicht akzeptieren. Umgekehrt akzeptiert der Automat „enthält kein ab“ alle Worte, die der Automat „enthält mindestens ein ab“ nicht akzeptiert. Wir erhalten also den Automaten zu „enthält kein ab“ aus dem Automaten „enthält mindestens ein ab“, indem wir die akzeptierenden und die nicht akzeptierenden Zustände tauschen. Dann sind q_0 und q_1 akzeptierend und q_2 ist eine Falle.

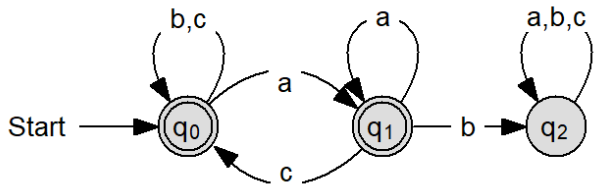

Abbildung 1.11: kein „ab“

Gesucht ist ein endlicher Automat, der genau ein „ab“ enthält. Um festzustellen, ob ein Muster in einer Zeichenfolge enthalten ist, muss man bis zwei zählen. Ist das Muster ein zweites Mal enthalten, dann ist man in einer Falle und braucht nicht weiter zu zählen, Der gesuchte Automat kann also zusammengesetzt werden aus zwei Automaten, die das Muster ab erkennen, und unserer Restschleife.

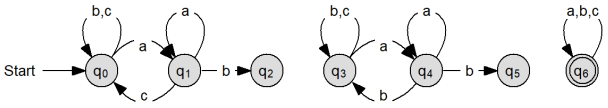

Abbildung 1.12: Bestandteile von genau ein „ab“

Die akzeptierenden Zustände beginnen, wenn das erste „ab“ abgeschlossen ist, und enden, wenn das zweite „ab“ fertig ist.

1.4 Teilbarkeit

Die Frage, ob bestimmte Zahlen durch eine andere Zahl teilbar ist, spielt eine wichtige Rolle in der Mathematik und in der Informatik. Dabei kann man zwei Arten der Lösung unterscheiden:

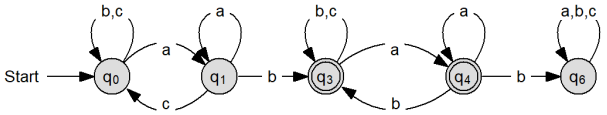

Abbildung 1.13: genau ein „ab“

1.4.1 Endziffern

Bestimmte Teiler lassen sich direkt an den Zahlen ablesen, vor allem an den Endziffern. So ist eine Zahl durch 5 teilbar, wenn die Endziffer 0 oder 5 ist. Die Teilbarkeit durch 4 lässt sich an den beiden letzten Ziffern ablesen, wenn man weiß, dass 100 oder alle Vielfachen von 100 durch 4 teilbar sind.

Ein Nachteil dieses Verfahrens ist, dass es nur für bestimmte Zahlen funktioniert und dass es nicht auf andere Zahlensysteme übertragbar ist. Z.B. lässt sich die Teilbarkeit durch 5 im Dualsystem nicht einfach an den Endziffern ablesen.

1.4.2 Division mit Rest

Zu einem anderen Lösungsansatz gelangt man über die ganzzahlige Division mit Rest und den bekannten Satz über die Teilbarkeit: „Ist a teilbar durch b, dann sind auch alle Vielfachen von a teilbar durch b.“

Die ganzzahlige Division mit Rest wird in der dritten Klasse behandelt vor der Einführung der Bruchzahlen. Sie funktioniert wie folgt:

- 1 durch 3 gleich 0 Rest 1
- 2 durch 3 gleich 0 Rest 2
- 3 durch 3 gleich 1 Rest 0
- 4 durch 3 gleich 1 Rest 1
- 5 durch 3 gleich 1 Rest 2
- 6 durch 3 gleich 2 Rest 0

Man sieht, dass bei der Division durch 3 nur die Reste 0, 1 und 2 auftauschen. Allgemein gilt,

bei der Division durch n treten dir Reste von 0 bis (n-1) auf. Wir benötigen also die Zustände 'Rest 0', 'Rest 1' und 'Rest 2'. Ausgehend vom Startzustand lassen sich die einstelligen Zahlen schnell diesen Resten zuordnen.

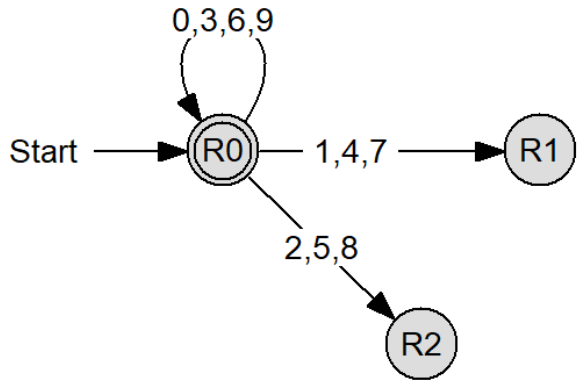

Abbildung 1.14: Startzustand Division durch Drei

Nun muss man überlegen, was passiert, wenn man im Dezimalsystem eine Ziffer hinten an eine Zahl anfügt: Die Zahl wird mit 10 multipliziert und dann die neue Ziffer addiert. Aus 13 wird durch Anfügen von 7 137. Der Rest 1 (bei Division durch 3) wird ebenfalls verzehnfacht und 1 addiert, also 17 bzw Rest 2. Wir gelangen also aus dem Zustand R1 durch Eingabe einer 7 in den Zustand R2. Entsprechend berechnet man die anderen Übergänge.

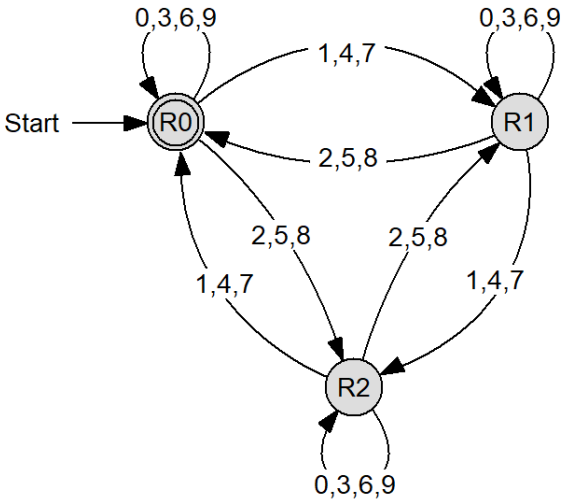

Abbildung 1.15: Division durch Drei im Dezimalsystem

Der Vorteil dieser Lösungsmethode mit den Resten ist, dass sie unabhängig vom Zahlensystem funktioniert. Auch im Dualsystem gibt es bei Division durch 3 nur die Reste 0, 1 und 2. Der entsprechende Automat hat also 3 Zustände. Beim Anfügen einer Null wird der Rest verdoppelt. Beim Anfügen einer Eins wird der Rest verdoppelt und dann Eins addiert.

1.5 Programmierung von endlichen Automaten

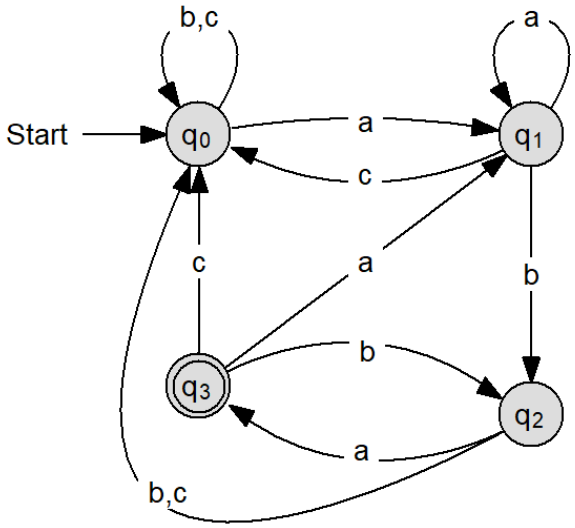

Abbildung 1.16: DEA für „endet auf aba“

Als Beispiel betrachten wir den endlichen Automaten mit dem Eingabealphabet {a, b, c}, der alle Worte erkennt, die auf „aba“ enden.

1.5.1 Erste Version des DEA

Für die Überführungsfunktion verwenden wir einen Reporter `Delta` (diesen Buchstaben gebraucht AutoEdit) mit den Parametern `Zustand` und `EZ` (Eingabezeichen). `Delta` liefert den Folgezustand.

`Delta` besteht überwiegend aus geschachtelten Verzweigungen. Zunächst müssen die vier Zustände unterschieden werden und dann das jeweilige Eingabezeichen.

Für die Eingabe verwenden wir ebenfalls einen Reporter. Das scheint jetzt etwas überdimensioniert, aber wir wollen in einem zweiten Schritt ungültige Eingaben herausfiltern und lagern die Eingabe deshalb aus:

Abbildung 1.17: Überführungsfunktion als geschachtelte Schleife

Abbildung 1.18: Eingabe-Block erste Fassung

Für unseren DEA benötigen wir nun noch eine Reihe globaler Variablen: das Eingabealphabet, die Zustandsmenge, die Menge der akzeptierenden Zustände, den (aktuellen) Zustand, das Wort, das überprüft werden soll, und eine Zählvariable i, weil wir das Wort Zeichen weise durchgehen müssen. Das Hauptprogramm setzt den Zustand q0 (Startzustand), und das Wort auf die eingegebene Zeichenfolge. Dann wird für jeden Buchstaben `letter i of wort` aus Zustand und Eingabezeichen der Folgezustand bestimmt. Zum Schluss überprüft das Programm, ob der aktuelle Zustand in der List der akzeptierenden Zustände enthalten ist und macht eine entsprechende Ausgabe:

In der aktuellen Fassung unseres DEA greifen wir noch nicht auf das Eingabealphabet und die Liste der Zustände zurück.

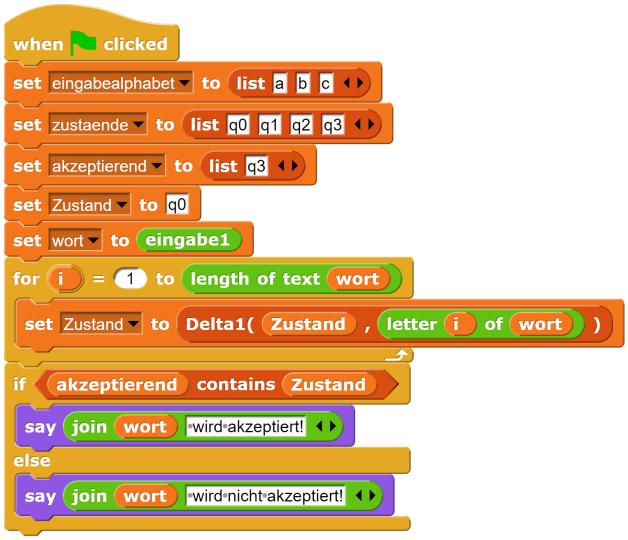

Abbildung 1.19: Hauptskript

1.5.2 Erste Verbesserung: Filterung der Eingabe

In der jetzigen Fassung ist der Automat nicht gegen Fehleingaben gesichert. Z.B. wird das Wort „a2b“ akzeptiert, obwohl 2 nicht zu den gültigen Eingabezeichen gehört. Nun könnte man in die Eingabe eine Abfrage einbauen, ob das Wort nur aus den Buchstaben a, b und c besteht. Wir wählen hier einen anderen Weg, der sich stärker an der Definition des DEA orientiert. Innerhalb unseres Reporters Eingabe gehen wir die Zeichenfolge einzeln durch und überprüfen, ob die eingegebenen Buchstaben zum Eingabealphabet gehören. Dafür gibt es den Block `liste contains thing`. Nur die zulässigen Zeichen werden in das Eingabewort aufgenommen.

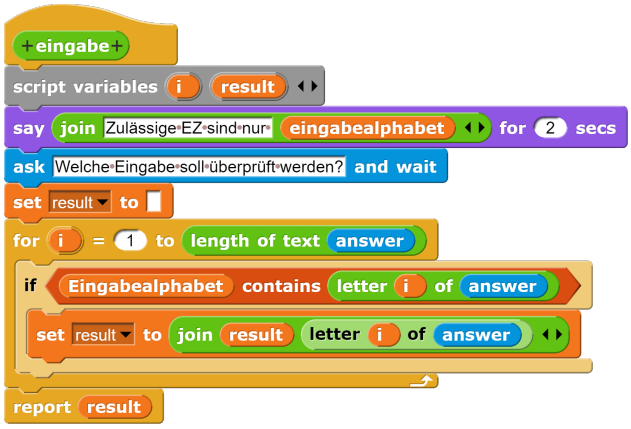

Abbildung 1.20: Eingabe-Block verbesserte Fassung

1.5.3 Zweite Verbesserung: Überführungsfunktion als Tabelle

Schon für unseren kleinen DEA wird die Überführungsfunktion Delta mit all den Fallunterscheidungen unübersichtlich. Das gilt erst recht, wenn wir mehr Zustände oder mehr Eingabezeichen haben. Eine viel übersichtlichere Darstellung der Überführungsfunktion findet sich in AutoEdit/Flaci.com als Tabelle:

δ	a	b	c
q_0	q_1	q_0	q_0
q_1	q_1	q_2	q_0
q_2	q_3	q_0	q_0
q_3	q_1	q_2	q_0

Tabelle 1.1: Überführungsfunktion DEA

Eine mögliche Lösung für Delta wäre also: Suche mit Hilfe des aktuellen Zustands die richtige Zeile und mit Hilfe des Eingabezeichens die richtige Spalte der Tabelle und gib den Zustand im Schnittpunkt von Zeile und Spalte als neuen Zustand aus.

Dazu benötigen wir den Index, an denen der Zustand in der Zustandsmenge und das Eingabezeichen im Eingabealphabet stehen. Dafür gibt es seit Snap! 6 den Block `index of element in liste`. Wenn wir zwei dieser Blöcke schachteln, indem wir den zweiten in den freien Platz für die Liste des ersten einsetzen, dann können wir auf ein Element in einer zweidimensionalen Reihung zugreifen, wobei der erste Index für die Spalte steht und der zweite für die Zeile:

Abbildung 1.21: Neue Fassung der Überführungsfunktion

Im Hauptskript benötigen wir eine zusätzliche globale Variable `delta`, in der wir die Tabelle der Überführungsfunktion speichern:

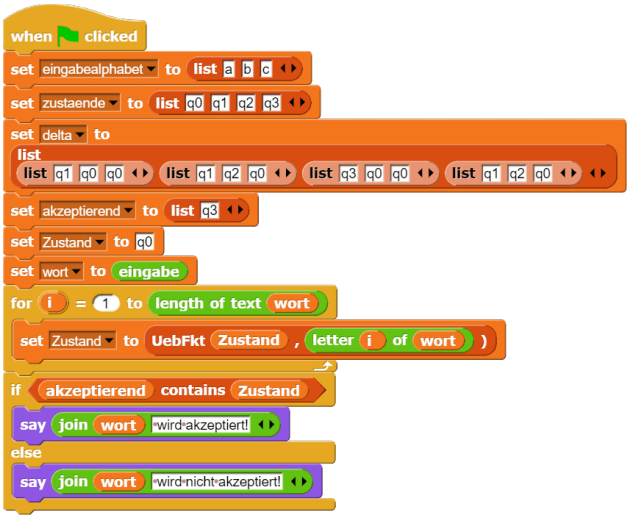

Abbildung 1.22: 2. Version des DEA

Diese Version des DEA lässt sich problemlos an einen anderen endlichen Automaten anpassen. Wir müssen lediglich das Eingabealphabet, die Zustandsmenge und die Menge der akzeptierenden Zustände anpassen und die Tabelle mit der neuen Überführungsfunktion einsetzen.

1.5.4 Dritte Verbesserung: Funktionen höherer Ordnung

Das Filtern der Eingabe lässt sich durch Verwendung der `Keep`-Funktion viel eleganter gestalten. Wir wandeln die Eingabe durch `split Wort by letter` in eine Liste um und behalten nur die Elemente, die im Eingabealphabet enthalten sind:

Abbildung 1.23: Filtern der Eingabe

Unsere Überführungsfunktion besteht ja nur aus einer einzelnen Zeile. Insofern ist die Funktion eigentlich überflüssig (zumal sie nur an einer Stelle aufgerufen wird). Wir können die geschachtelten `item of Index in Liste`-Blöcke auch direkt in die Zuweisung einsetzen.

Die `FOR`-Schleife können wir durch einen `for each item in Liste`-Block ersetzen, wobei wir dann in der Zuweisung `item` statt `EZ` als Spaltenindex verwenden müssen.

Damit erhalten wir die dritte Fassung unseres deterministischen endlichen Automaten:

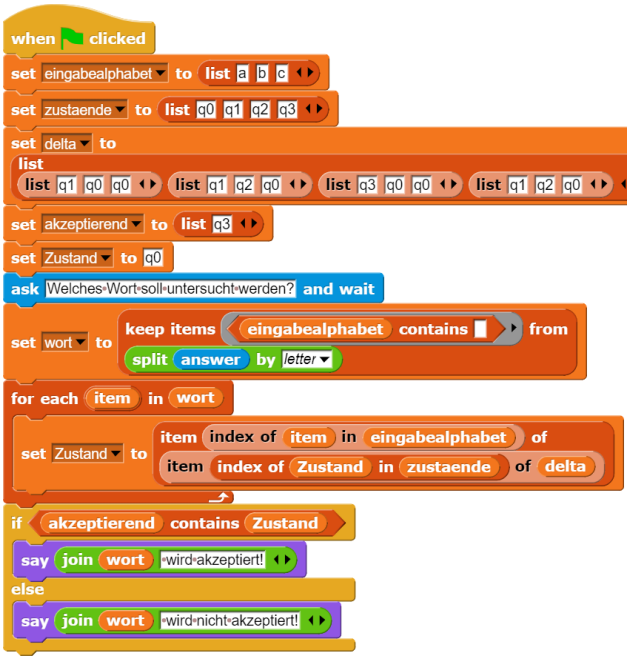

Abbildung 1.24: 3. Version des DEA

1.6 Aufgaben

Aufgabe 1.3 *Gegeben sei das Eingabealphabet $E = \{a, b, c\}$. Wähle einen Buchstaben aus und konstruiere jeweils das Zustandsdiagramm zu folgenden Aufgaben:*

1. *Akzeptiert werden alle Worte, die den Buchstaben mindestens ein Mal enthalten.*
2. *Akzeptiert werden alle Worte, die mit dem Buchstaben beginnen.*
3. *Akzeptiert werden alle Worte, die mit dem Buchstaben enden.*
4. *Akzeptiert werden alle Worte, die den Buchstaben nicht enthalten.*
5. *Akzeptiert werden alle Worte, die den Buchstaben genau ein Mal enthalten.*

Aufgabe 1.4 *Gegeben sei das Eingabealphabet $E = \{a, b, c\}$. Wähle ein Muster aus zwei Buchstaben aus und bearbeite die Aufgaben wie unter 2.4, wobei das Muster an die Stelle des Buchstabens in der Aufgabenstellung tritt. Es können verschiedene oder gleiche Buchstaben sein.*

Aufgabe 1.5 *Gegeben sei das Eingabealphabet $E = \{a, b, c\}$. Wähle ein Muster aus drei Buchstaben aus und bearbeite die Aufgaben wie unter 2.4, wobei das Muster an die Stelle des Buchstabens in der Aufgabenstellung tritt. Es können verschiedene oder gleiche Buchstaben vorkommen.*

Aufgabe 1.6 *Gegeben sei das Eingabealphabet $E = \{0, 1\}$. Konstruiere das Zustandsdiagramm eines DEA, der alle Dualzahlen akzeptiert, die durch 4 teilbar sind.*

Aufgabe 1.7 *Gegeben sei das Eingabealphabet $E = \{0, 1\}$. Konstruiere das Zustandsdiagramm eines DEA, der alle Dualzahlen akzeptiert, die durch 5 teilbar sind.*

Aufgabe 1.8 *Gegeben sei das Eingabealphabet $E = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Konstruiere das Zustandsdiagramm eines DEA, der alle Dezimalzahlen akzeptiert, die durch 4 teilbar sind.*

Aufgabe 1.9 *Erläutere, was man unter der Vollständigkeit und unter der Determinierbarkeit eines endlichen Automaten versteht. Untersuche den folgenden Automaten auf Vollständigkeit und Determinierbarkeit:*

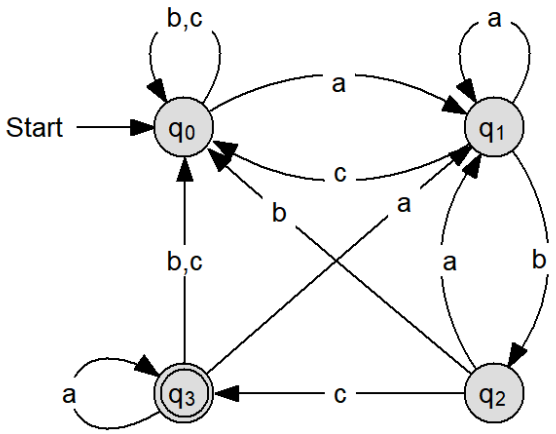

Abbildung 1.25: Beispielautomat

2 Mealy-Automaten

In diesem Kapitel lernen Sie

- was die Unterschiede zwischen deterministischen endlichen Automaten (DEA) und Mealy-Automaten sind
- wie man Mealy-Automaten konstruiert.

2.1 Definition und Funktionsweise

Deterministische endliche Automaten (DEA) dienen allgemein der Überprüfung von Zeichenketten, also: wurde ein gültiger Zugangscode eingegeben oder ist das Computerprogramm „richtig“, d.h. wurden die Syntaxregeln der Programmiersprache eingehalten?

Für andere Zwecke wäre es schön, wenn man mit einem Automaten etwas steuern könnte, z.B. eine Ampel, einen Getränkeautomaten einen Fahrstuhl oder eine myoelektrische Armprothese. Wir ergänzen deshalb unseren Automaten um ein Ausgabealphabet und eine Ausgabefunktion, die jedem Zustand und jedem Eingabezeichen ein Ausgabezeichen zuordnet. Dafür lassen wir die akzeptierenden Zustände weg. Ein **Mealy-Automat** besteht also aus sechs Teilen:

- einer endlichen Menge von Zuständen Q
- einem Eingabealphabet Σ,
- einem Ausgabealphabet Δ,
- einer Überführungsfunktion, die zu Zustand und Eingabezeichen den Folgezustand berechnet,
- einer Ausgabefunktion, die zu Zustand und Eingabezeichen das Ausgabezeichen berechnet, und
- einem Startzustand (hier q_0),

Auch ein Mealy-Automat verfügt über ein Eingabeband, das nacheinander die Eingabezeichen liefert. Dazu kommt jetzt noch ein Ausgabeband, auf dem die jeweilige Ausgabe notiert wird.

Vergleicht man deterministische endliche Automaten und Mealy-Automaten, so stellt man fest: beide Arten verfügen über eine Zustandsmenge, ein Eingabealphabet, eine Überführungsfunktion und einen Startzustand. Nur der deterministische endliche Automat hat eine Menge von akzeptierenden Zuständen. Nur der Mealy-Automat hat ein Ausgabealphabet und eine Ausgabefunktion.

Ein Mealy-Automat hat ein Eingabeband, auf dem Zeichen aus dem Eingabealphabet stehen. Er befindet sich zu Beginn im Startzustand. In jedem Schritt liest er genau ein Eingabezeichen ein. Er berechnet aus dem aktuellen Zustand und dem Eingabezeichen mit Hilfe der Ausgabefunktion des Ausgabezeichen und mit Hilfe der Überführungsfunktion den Folgezustand. Dann schreibt er das Ausgabezeichen auf das Ausgabeband, rückt dieses eine Stelle vor und merkt sich den neuen Zustand.

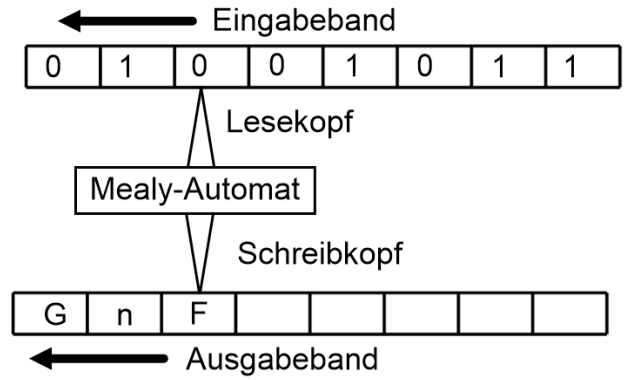

Abbildung 2.1: Aufbau eines Mealy-Automaten

Auch einen Mealy-Automaten kann man sich als Computer mit nur einem Speicherplatz vorstellen. Der Inhalt dieses Speichers ist der Zustand, in dem sich der Automat befindet. Die Ausgaben, z.B. Fahrkarte beim Fahrkartenau-

tomaten oder Saft beim Getränkeautomaten sind keine Zustände.

2.2 Fahrkartenautomat

Ein Fahrkartenautomat gibt Fahrkarten zum Wert von 2 Euro aus. Er verfügt über die zwei Tasten „Fahrkarte“ und „Geldrückgabe“. Der Automat akzeptiert nur 1 Euro- und 2 Euro- Stücke. Damit besteht das Eingabealphabet E aus den Zeichen F(ahrkarte), (Geld)R(ückgabe), 1 (Ein Euro) und 2 (Zwei Euro). Eine Überzahlung soll möglichst verhindert werden, indem überzähliges Geld sofort wieder zurückgegeben wird.

Bei Automaten, bei denen man etwas bezahlen muss, gibt es grundsätzlich zwei Möglichkeiten: Bei den meisten Getränkeautomaten wird der Betrag an, den man bisher eingegeben hat, angezeigt. In Abhängigkeit vom Betrag kann man dann Dinge kaufen. Die andere Art gibt es häufig in Parkhäusern. Steckt man das Ticket ein, so wird der Betrag angezeigt, der noch zu entrichten ist.

Die Zustände unseres Fahrkartenautomaten sind der Geldbetrag, den er enthält, also 0 Euro, 1 Euro und 2 Euro. Die Ausgaben sind 1 Euro, 2 Euro, F(ahrkarte) und n(ichts).

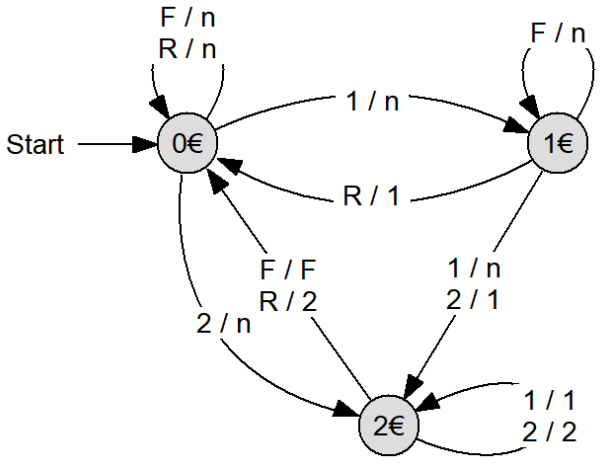

Abbildung 2.2: Fahrkartenautomat

Wenn der Automat im Zustand 0€ ist, also kein Geld enthält, und die Fahrkartentaste oder die Geldrückgabetaste gedrückt wird, bleibt der Automat im Zustand 0€ und gibt n aus. Wenn der Automat im Zustand 0€ ist und 1 Euro wird eingeworfen, geht er in den Zustand 1€ Euro und gibt nichts aus. Wenn der Automat im Zustand 0€ ist und 2 Euro werden eingeworfen, dann geht er in den Zustand 2€ Euro und gibt nichts aus.

Im Zustand 1€ (d.h. bisher 1 Euro eingeworfen) verhält sich der Automat wie folgt: Bei Drücken der F-Taste gibt der Automat nichts aus und bleibt im Zustand 1€. Bei Drücken der Geldrückgabe-Taste gibt der Automat 1 Euro zurück und geht in den Zustand 0€. Bei Einwurf von 1 Euro geht er in den Zustand 2€ und gibt nichts aus.

Im Zustand 2€ gibt der Automat beim Drücken der Fahrkarten-Taste eine Fahrkarte aus und geht in den Zustand 0€. Beim Drücken der Geldrückgabe-Taste gibt er zwei Euro zurück und geht ebenfalls in den Zustand 0€. Bei Eingabe von 1 Euro oder 2 Euro gibt er das eingeworfene Geld zurück, um eine Überzahlung zu vermeiden, und bleibt im Zustand 2€.

2.3 Aufgaben

Aufgabe 2.1 *Beschreiben Sie die Steuerung einer **Verkehrsampel** als Mealy-Automaten. Eingabezeichen sind die Signale F (Farbe halten) und W (Farbe wechseln). Ausgabezeichen sind die Farbkombinationen der Ampel: rot, rot-gelb, grün, gelb, die dem zuletzt eingenommenen Zustand entsprechen sollen.*

Aufgabe 2.2 *Erweitern Sie die **Ampel** um eine Bedarfssteuerung. Das Eingabezeichen R (verlängerte Rotphase) bewirke, dass die Rotphasen der Ampel doppelt so lang sind wie die Grünphasen. Beim Eingabezeichen G (verlängerte Grünphase) reagiere die Ampel entsprechend. Lassen Sie die Eingabezeichen F und W aus Aufgabe 2.11 weg (die Ampel enthalte jetzt einen automatischen Takt).*

Vorbemerkung zu den Aufgaben 2.3 bis 2.7: Bei Automaten, an denen man etwas bezahlen kann, gibt es zwei unterschiedliche Konstruktionsweisen. Entweder man wirft so lange Geld ein, bis der benötigte Betrag erreicht

ist, und kann dann mit der passenden Wahltaste die Ausgabe veranlassen. Die Zustände symbolisieren dann jeweils den Geldbetrag, der noch zu zahlen ist. Oder man wählt ein Angebot (z.B. eine Fahrkarte) und der Automat zeigt dann an, wie viel Geld noch zu zahlen ist. Die Zustände symbolisieren dann die jeweilige Bestellung und den Restbetrag.

Aufgabe 2.3 *Entwickeln Sie das Zustandsdiagramm eines* ***Getränkeautomaten*** *mit nur einer Sorte Fruchtsaft. Eingabezeichen sind G (Geld), W (Wahl) und R (Geldrückgabeknopf). Ausgegeben werden S (Saft), G (Geld) oder N (nichts).*

Aufgabe 2.4 *Entwickeln Sie das Zustandsdiagram eines* ***Kaffee-/Tee-Automaten****, der sowohl 50-Cent-Stücke wie 1-€-Stücke annimmt. Weitere Eingaben sind R (Geldrückgabe), K (Kaffee) und T (Tee). Der Preis für eine Tasse betrage 1,50 €. Überzahlte Beträge werden nur insoweit einbehalten, dass man für 2 € ein Getränk für 1,50 € kaufen kann.*

Aufgabe 2.5 *Konstruieren Sie das Zustandsdiagramm eines* ***Parkhaus-Automaten****. Im Startzustand sind die möglichen Eingaben S1 (Parkschein bis 1 Stunde Parkzeit), S2 (Parkschein bis 2 Stunden Parkzeit) und S3 (Parkschein über 2 Stunden Parkzeit). Die Geldeingabe ist im Startzustand geschlossen. Wenn Sie einen Parkschein eingegeben haben, ändert sich das Eingabealphabet: Sie können keinen zweiten Parkschein eingeben, sondern nur noch Geld: F (50 Cent), E (1 Euro) und Z (Zwei Euro). Nach Eingabe des Parkscheins wird der jeweilige Restbetrag angezeigt. Das Parken bis zu einer Stunde kostet 1 Euro, bis zu 2 Stunden 2 Euro und über zwei Stunden 4 Euro. Ausgaben sind n (nichts) und P (Parkschein bezahlt).*

Aufgabe 2.6 *Ein* ***Abspielgerät für MP3-Dateien*** *hat insgesamt drei Tasten: In der Mitte eine Taste für das Starten und Stoppen, sowie links und rechts eine Vor-/Zurück-Taste, um den nächsten oder vorhergehenden Titel auszuwählen. Im Stopp-Zustand ist die Vor-/Zurück-Taste außer Funktion. Beschreiben Sie mit Hilfe eines Zustandsdiagramms die Funktionsweise eines solchen Gerätes unter der vereinfachenden Annahme, es gäbe nur vier Titel.*

Beispielwörter	Ausgabe
1	0
0010	0001
1010	0101
1000111	0100011

Aufgabe 2.7 *Eine Dualzahl soll um eine Stelle nach rechts verschoben werden (right shift). Dies entspricht einer Division der Zahl durch 2, wobei die Anzahl der Stellen gleich bleiben soll. Entwerfen Sie einen Mealy-Automaten, der die jeweils eingegebene Zahl dementsprechend verarbeitet. Eingaben sind 0, 1, Lz (Leerzeichen):*

Aufgabe 2.8 *Beschreiben Sie eine* ***Fahrstuhlsteuerung*** *als endlichen Automaten. Angefahren werden die drei Stockwerke E (Erdgeschoss), 1 (1. Stock) und 2 (2. Stock).*

Erläutern Sie, warum es sinnvoll ist, bei den Zuständen nicht nur zwischen den Stockwerken, sondern auch zusätzlich zwischen der Fahrtrichtung zu unterscheiden. Denken Sie dabei an ein Gebäude mit mehr Stockwerken.

Als Eingaben erhält das System die Anforderungssignale aus den drei Stockwerken als dreistellige Dualzahlen. Dabei steht die erste Ziffer für die Anforderungen aus dem 2. Stock, die mittlere Ziffer für die Anforderungen aus dem 1. Stock und die letzte Ziffer für die Anforderungen aus dem Erdgeschoss. 101 bedeutet also: Es liegen Anforderungen aus dem 2. Stock und aus dem Erdgeschoss vor. Hält der Fahrstuhl in einem Stockwerk, so werden die Anforderungen aus diesem Stockwerk gelöscht, sobald die Fahrstuhltür wieder schließt.

Höchste Priorität hat immer die Anforderung aus dem Stockwerk, in dem sich der Fahrstuhl gerade befindet. Liegen Anforderungen aus einem höheren und einem niedrigeren vor, so hat die Anforderung Priorität, die in der momentanen Fahrtrichtung des Fahrstuhls liegt. Nach Erreichen des nächsten Stockwerks hält der Fahrstuhl und die Anlage überprüft erneut die Anforderungen.

3 Automaten und reguläre Sprachen

In diesem Kapitel lernen Sie

- welche Eigenschaften formale Sprachen im Vergleich zu natürlichen Sprachen haben,
- wie man die von einer Grammatik erzeugte Sprache beschreibt,
- wie man reguläre Grammatiken für formale Sprachen entwirft,
- wie man eine reguläre Grammatik in einen endlichen Automaten überführt und umgekehrt,

3.1 Sprache und Grammatik

Rechner arbeiten im Prinzip mit Texten, d.h. mit Folgen von Symbolen aus einem bestimmten Alphabet. Auch blockbasierte Programmiersprachen werden letztlich als Zeichenfolgen gespeichert (Snap! z.B. im XML-Format). Ein- und Ausgaben können als Texte dargestellt werden. Die Grundbegriffe, die hier eingeführt werden, sind **Alphabet**, **Wort** und **Sprache**.

Eine endliche nichtleere Menge Σ heißt **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben**, Zeichen oder Symbole genannt. Wir verwenden den Begriff Alphabet genau so wie bei natürlichen Sprachen, wobei wir nicht nur herkömmliche Buchstaben, sondern beliebige Zeichen verwenden können.

Beispiele wichtiger Alphabete:

- $\Sigma_{bool} = \{0, 1\}$ ist Alphabet der Dualziffern, mit dem alle Rechner arbeiten.
- $\Sigma_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist das Alphabet der Dezimalziffern.
- $\Sigma_{lat} = \{a, b, c, \ldots, z\}$ ist das lateinische Alphabet.
- $\Sigma_{logic} = \{0, 1, x, \{, \}, \wedge, \vee, \neg\}$ ist das Alphabet, mit dem man logische Formeln darstellen kann.

Unter einem **Wort** über einem Alphabet Σ verstehen wir in der Informatik eine endliche (eventuell leere) Folge von Buchstaben aus Σ. Das **leere Wort** ϵ ist die leere Buchstabenfolge. In der deutschsprachigen Literatur findet sich für ϵ teilweise auch die Bezeichnung λ. ϵ ist niemals in einem Alphabet enthalten. Es handelt sich ja gerade um das Wort, welches ohne jedes Symbol aus einem Alphabet gebildet wird.

Als **Länge** $|w|$ eines Wortes bezeichnen wir die Anzahl seiner Buchstaben. Wenn wir die Häufigkeit des Vorkommens eines bestimmten Buchstabens beschreiben wollen, setzen wir den Buchstaben als Index dahinter. $|w|_0$ wäre die Anzahl der Nullen im Wort w.

Σ^* ist die Menge aller Wörter über dem Alphabet Σ, das leere Wort eingeschlossen. Die Menge aller nichtleeren Wörter über dem Alphabet Σ ist Σ^+.

Eine **formale Sprache** L über einem Alphabet Σ ist eine Teilmenge von Σ^*, also eine beliebige Menge von Wörtern aus einem Alphabet.

Wir sprechen in den Informatik von formalen Sprachen, um sie von den natürlichen Sprachen zu unterscheiden. Natürliche Sprachen sind immer eingebettet in einen Kommunikationszusammenhang, in dem über Tonfall, Gestik und Mimik auch Informationen übermittelt werden. Dadurch entsteht eine gewisse Unschärfe, die für natürliche Sprachen charakteristisch ist. Bestimmte Charakteristika natürlicher Sprachen wie z.B. Wortspiele nutzen genau diese Unschärfe aus. In der Informatik möchten wir aber Algorithmen in eindeutiger und für den Computer verständlicher Weise formulieren. Wir legen bei den formalen Sprachen den Schwerpunkt auf die Syntax, d.h. die Regeln, nach denen Wörter gebildet werden. Die Semantik, d.h. die Bedeutung der Wörter, spielt zunächst einmal keine Rolle.

Deterministische endliche Automaten (DEA) haben wir in Klasse 11 als Akzeptoren ein-

geführt. Sie sind also dafür geeignet, zu erkennen, ob ein gegebenes Wort zu einer bestimmten Sprache gehört. Endliche formale Sprachen kann man durch Aufzählung oder Aufschreiben spezifizieren. Auch die übrigen Automaten (NEA, Kellerautomaten und Turingmaschinen) sind Akzeptoren. Grammatiken bieten einen anderen Weg, um unendliche Sprachen durch endliche Systeme eindeutig zu beschreiben: **Grammatiken sind Systeme zur Erzeugung von Wörtern**. In der Informatik kommt eine wichtige Bedeutung hinzu: **Kontextfreie Grammatiken**, die wir in diesem Kapitel kennen lernen, benutzt man, um **Programmiersprachen** darzustellen. Alternativ ist auch eine Notation mit Hilfe von Syntaxdiagrammen möglich. So entsprechen Worte, die durch spezielle kontextfreie Grammatiken erzeugt wurden, syntaktisch korrekten Programmen in den jeweils modellierten Programmiersprache.

Die **Syntax** einer formalen Sprache gibt an, wie die Elemente der Sprache, also Sätze oder Worte, zusammengesetzt sind, d.h. welche Form sie haben. Syntaktische Gebilde sind deshalb Zeichenreihen, Diagramme, Programme, Terme, Formeln und Spezifikationen. Die **Semantik** gibt den Elementen einer formalen Sprache eine Bedeutung. Semantische Gebilde sind deshalb Zahlen, Mengen, Listen, Funktionen und Datenstrukturen.

Zu einer Grammatik gehören zwei Alphabete, eine Alphabet aus Variablen, die in den Wörtern der generierten Sprache nicht auftreten dürfen, und ein Alphabet aus **Terminalzeichen (oder Buchstaben**), die die Elemente der Sprache bilden. **Nichtterminale** oder Wörter, die Nichtterminalsymbole enthalten, dürfen durch andere Wörter ersetzt werden. Wie und was ersetzt werden soll, wird durch die Produktionen oder Regeln bestimmt. Man könnte sagen, dass eine Grammatik eine Menge von Regeln zur Überführung einer Menge von Nichtterminalzeichen in eine Wort aus Terminalzeichen ist.

Formale Sprachen werden also durch ihre Grammatik definiert. Eine Grammatik G ist ein Viertupel { V, Σ, P, S }. Sie besteht aus

- der endlichen Variablenmenge V (auch Nichtterminale genannt und dann als Menge mit NT bezeichnet),
- dem endlichen Alphabet der Terminalzeichen Σ, wobei Σ keines der Nichtterminale enthalten darf,
- der endlichen Menge P von Produktionen (Regeln) und
- der Startvariablen S mit $S \in$ V.

3.2 Chomsky-Klassifizierung

Der amerikanische Linguist Noam Chomsky hat ein Regelwerk aufgestellt, mit dessen Hilfe sich Grammatiken formaler Sprachen in vier Gruppen einteilen lassen, wobei die erste Gruppe die umfassendste ist und jede weitere Gruppe in den vorigen enthalten ist.

- **Typ-0-Grammatiken oder allgemeine Chomsky-Grammatiken**. Die zugehörigen Ersetzungsregeln sind der Form $\alpha \rightarrow \beta$, d.h. links und rechts dürfen beliebige Folgen von Variablen (Nichtterminalsymbolen) und Terminalsymbolen stehen. Für diese Grammatiken gelten keine weiteren Einschränkungen. Den allgemeinen Chomsky-Grammatiken entsprechen als Automaten die **Turingmaschinen**.
- **Typ-1-Grammatiken oder kontextsensitive Grammatiken**. Die zugehörigen Ersetzungsregeln sind der Form $\alpha A \nu \rightarrow \alpha \beta \nu$. Das bedeutet: Wenn A im Kontext von α und ν steht, kann es durch eine beliebige nichtleere Folge von terminalen und nichtterminalen Symbolen ersetzt werden. Für diese Grammatiken gilt, dass man kein Teilwort α durch ein kürzeres Teilwort β ersetzen kann.
- **Typ-2-Grammatiken oder kontextfreie Grammatiken**. Die zugehörigen Ersetzungsregeln sind der Form $A \rightarrow \beta$, d.h. ein Nichtterminalsymbol wird ohne Berücksichtigung des Kontextes, in dem es steht, durch eine beliebige nichtleere Folge von terminalen und nichtterminalen Symbolen ersetzt. Den kontextfreien

Grammatiken entsprechen die nichtdeterministischen **Kellerautomaten**.

- **Typ-3-Grammatiken oder reguläre Grammatiken**. Alle Ersetzungsregeln sind entweder der Form $A \to a$ oder $A \to Ba$. Das bedeutet, entweder wird ein Nichtterminalsymbol durch ein Terminalsymbol ersetzt oder ein Nichtterminalsymbol wird durch eine Kombination aus Nichtterminalsymbol und Terminalsymbol ersetzt, wobei das Nichtterminalsymbol bei den linksregulären Sprachen links steht. Insbesondere die rekursive Regel $A \to Aa$ ist linksregulär. Daneben gibt es auch rechtsreguläre Sprachen, bei denen das Nichtterminalsymbol rechts steht: $A \to aB$. Den regulären Grammatiken entsprechen die **endlichen Automaten**.[1] Die Ersetzungsregel $A \to a$ bezeichnen wir als **terminierende Regel**. Sie beendet eine Folge von Nichtterminalsymbolen und setzt dafür ein Terminalsymbol ein, also einen Buchstaben, der nicht weiter ersetzt wird.

3.3 Endliche Automaten als Programm

Wir haben uns in Kapitel 1 mit den Endlichen Automaten beschäftigt und dabei vor allem die Darstellung Endlicher Automaten als Zustandsdiagramm betrachtet. Zustandsdiagramme sind eine sehr anschauliche Darstellung Endlicher Automaten. In diesem Kapitel wollen wir eine andere Form der Darstellung von Endlichen Automaten vorstellen, nämlich als Programm.

Ein endlicher Automat hat keinen Speicher zur Verfügung außer dem Speicher, in dem das Programm gespeichert wird und dem Zeiger, der auf die aktuelle Zeile des Programms zeigt. Das Programm eines Endlichen Automaten verwendet also keine Variablen. Das erscheint zunächst überraschend, dann in den Grundlagen der Algorithmik haben wir gelernt, dass man mit Variablen rechnet (etwas anders ist es mit Funktionen höherer Ordnung, aber das soll zunächst außer Betracht bleiben). Die einzige Information, die sich im Programmablauf ändert, ist die Position des Zeigers, die die aktuelle Zeile des Programms beschreibt.

Das Programm wird zusätzlich eingeschränkt dadurch, dass wir für Endliche Automaten nur einen einzigen Operationstyp zulassen. Sei $\Sigma = \{a_1, a_2, \ldots, a_k\}$ die Menge der Eingabezeichen. Dann darf der Automat nur die folgende Operation benutzen:

```
select input = a_1 goto i_1
       input = a_2 goto i_2
             ...
       input = a_k goto i_k
```

Das bedeutet, dass das Programm aus den verschiedenen Eingabezeichen das aktuelle aussucht und in Abhängigkeit von diesem Eingabezeichen und dem aktuellen Zustand (bzw. der aktuellen Programmzeile) den Wert des Zeigers auf die nächstes Programmzeile bestimmt. Der SELECT-Befehl überprüft das gesamte Eingabealphabet. Das Programm startet immer im Zustand 0.

Wenn es nur zwei Eingabezeichen gibt, z.B. $\{a, b\}$ oder $\{0, 1\}$, dann kann man statt des SELECT-Befehls, der aus verschiedenen Eingabezeichen eines heraussucht, auch einfach eine IF-ELSE-Verzweigung benutzen

```
if input = 1 then goto i else goto j
```

oder bei $\Sigma = \{a, b\}$

```
if input = a then goto i else goto j
```

Wir stellen das Programm für den in Abbildung 10.1. dargestellten Automaten auf:

```
0: if input = 1 goto 1 else goto 3
1: if input = 1 goto 0 else goto 2
2: if input = 1 goto 3 else goto 1
3: if input = 1 goto 2 else goto 0
```

Solche Programme benutzt man, um Entscheidungsprobleme zu lösen. Man wählt eine Teilmenge der Zeilen aus (die akzeptierenden Zustände). Wenn sich das Programm nach Abarbeitung des Eingabewortes in einer dieser Zeilen befindet, wird das Wort akzeptiert. Die Menge aller akzeptierten Wörter ist die von dem Programm akzeptierte Sprache.

[1] In der Literatur finden sich teilweise auch eine Definition regulärer Grammatiken, bei denen auf der rechten Seite der Ersetzungsregeln links oder rechts nicht nur ein Terminalzeichen, sondern eine Folge von Terminalzeichen stehen kann. Diese Definition führt auf die gleiche Klasse von Sprachen.

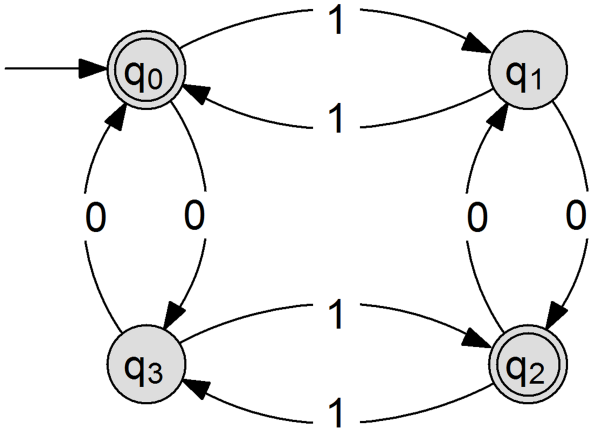

Abbildung 3.1: DEA für alle Worte, bei denen die Anzahl der Nullen und Einsen beides gerade oder beides ungerade ist

Diese Form der Darstellung eines Endlichen Automaten ist nicht so elegant wie die als Zustandsdiagramm, aber sie veranschaulicht die Art und Weise der Informationsverarbeitung durch einen Endlichen Automaten.

3.4 Reguläre Sprachen

Reguläre Sprachen gibt es in zwei Varianten. Wenn alle Ersetzungsregeln in der Form $A \to a$ oder $A \to Ba$ sind, liegt eine linksreguläre Grammatik vor, weil das Nichtterminalzeichen links steht. Wenn alle Ersetzungsregeln in der Form $A \to a$ oder $A \to aB$ sind, liegt eine rechtsreguläre Grammatik vor, weil das Nichtterminalzeichen rechts steht. In beiden Fällen kann die rekursive Regel $A \to Aa$ bzw $A \to aA$ zur Grammatik gehören.

Wenn wir den Zusammenhang zwischen regulären Sprachen und endlichen Automaten beschreiben sollen, müssen wir nächst definieren, was wir unter der Sprache eines endlichen Automaten verstehen. Wir definieren deshalb wie folgt: „Alle Eingabefolgen, die einen Automaten A in einen akzeptierenden Zustand überführen, bilden die Sprache $L(A)$ dieses endlichen Automaten." Mit dieser Definition der Sprache eines endlichen Automaten können wir den **Äquivalenz-Satz** für endliche Automaten und reguläre Grammatiken formulieren:

„Zu jedem endlichen Automaten mit einem akzeptierenden Zustand gehört eine linksreguläre Grammatik, die die gleiche Sprache wie der Automat akzeptiert, und umgekehrt."

Es gibt Möglichkeiten, die Einschränkung auf **einen** akzeptierenden Zustand zu umgehen. So könnte man alle akzeptierenden Zustände mit Epsilon-Übergängen zusammenfassen. Das dürfte allerdings über den Abiturstoff hinausgehen. Wir wollen den Äquivalenz-Satz beweisen, indem wir ein Verfahren angeben, mit dessen Hilfe die Umsetzung von endlichen Automaten in linksreguläre Grammatiken und umgekehrt geschehen kann. Dazu vergleichen wir zunächst die Bestandteile von Automaten und Grammatiken:

linksreguläre Grammatik	endlicher Automat
Terminalzeichen oder Symbole	Eingabealphabet Σ
Nichtterminal-zeichen oder Variable	Zustandsmenge (ohne den Start-zustand q_0)
Startsymbol S	akzeptierender Zustand q_e
Regelsystem R	Überführungs-funktion δ

In einem Zustandsdiagramm geht man vom Startzustand je nach Eingabezeichen über die Kanten bis zum akzeptierenden Zustand. Dabei wird das akzeptierte Wort von links nach rechts aufgebaut, indem auf jeder Kante **hinten** ein Buchstabe angefügt wird. Eine zulässige Folge von Eingabezeichen überführt einen endlichen Automaten vom Start- in den akzeptierenden Zustand.

Ein von einer Grammatik akzeptiertes Wort besteht aus Terminalsymbolen, die die lexikalischen Bestandteile des Wortes bilden. Eine linksreguläre Grammatik fügt bei jeder Anwendung einer Regel **vorn vor den bereits abgeleiteten Buchstaben** einen neuen Buchstaben (Terminalsymbol) ein, während ganz links das neue Nichtterminalsymbol steht. Dazu ein Beispiel: Wir haben als Zwischenschritt das Wort Abc abgeleitet und wollen darauf die Regel $A \to Ba$ anwenden. Dann ergibt sich

$Abc \rightarrow Babc$.

Die Richtung der Kanten im Zustandsgraph ist bei linksregulären Grammatiken also entgegengesetzt zur Richtung der Grammatikregeln. Da sich Regelsystem (der Grammatik) und Überführungsfunktion (des endlichen Automaten) entsprechend, liegt es nahe, eine Zuordnung zwischen den Ersetzungsregeln und den Kanten des Zustandsgraphen zu schaffen:

1. Umsetzungsregel: Im Zustandsgraph treten Kanten auf, die bei Eingabe von a aus dem Zustand B in den Zustand A führen. In der regulären Grammatik schrieben wir dies $A \rightarrow Ba$ oder in Worten: Folgezustand → alter Zustand Eingabezeichen.

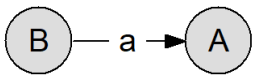

Abbildung 3.2: $A \rightarrow Ba$

2. Umsetzungsregel: Im Zustandsgraph treten Kanten auf, die bei Eingabe von a vom Startzustand q0 in den Zustand A führen. In der regulären Grammatik schreiben wir dies $A \rightarrow a$ oder in Worten: Folgezustand → Eingabezeichen.

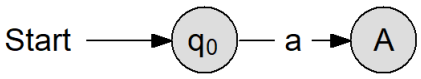

Abbildung 3.3: $A \rightarrow a$

Jede reguläre Grammatik benötigt **mindestens eine terminierende Regel der Form** $A \rightarrow a$. Wenn eine Grammatik nur Regeln der Form $A \rightarrow Ba$ enthält, lassen sich keine Worte ableiten, weil immer ein Nichtterminalsymbol stehen bleibt. Worte bestehen aber vollständig aus Terminalzeichen. **Hinweis**: Wenn Sie eine Grammatik aufstellen sollen, prüfen Sie, ob Sie ein beliebiges Wort mit dieser Grammatik bilden können. Wenn Sie keine Regel der Form $A \rightarrow a$ aufgenommen haben, können Sie kein Wort bilden, weil immer Nichtterminalsymbole in der Zeichenkette übrig bleiben.

3. Umsetzungsregel: Im Zustandsgraph treten Kanten auf, die bei Eingabe von a vom einem Zustand B in den akzeptierenden Zustand S führen (S ist das Startsymbol der Grammatik): In der regulären Grammatik schreiben wir dies $S \rightarrow Ba$ oder in Worten: Startsymbol → alter Zustand Eingabezeichen.

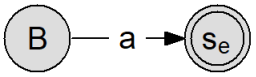

Abbildung 3.4: $S \rightarrow Ba$

Unabhängig von der Zahl der akzeptierenden Zustände ist es möglich, einen endlichen Automaten mit einem Startzustand in eine **rechtsreguläre Grammatik** umzuwandeln. Durch die Grammatik wird das Wort dann von „vorn“ aufgebaut. Terminierende Regeln sind dann diejenigen, die in einen akzeptierenden Zustand führen.

Ein Tipp: Wir schreiben in der Regel Zustände mit Index, z.B. q_0, q_1, q_2 usw. Es empfiehlt sich, bei der Umsetzung eines DEA in eine Grammatik die Zustände (wie in der Grammatik vorgeschrieben) mit großen Buchstaben zu bezeichnen. Handschriftlich lassen sich Indices und kleine Buchstaben nicht mehr unterscheiden.

3.4.1 DEA in linksreguläre Grammatik

Wir führen das Verfahren am einem Beispiel durch, das gleich auf ein Problem führt. Der folgende DEA erkennt alle Worte, die auf „10“ enden:

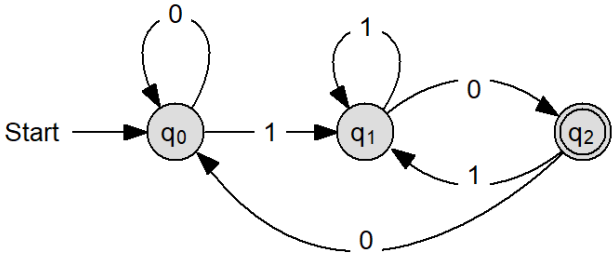

Abbildung 3.5: DEA für alle Worte, die auf 10 enden

Wir ersetzen die Bezeichnungen für die Zustände durch Großbuchstaben, wobei der akzeptierende Zustand die Bezeichnung S erhält:

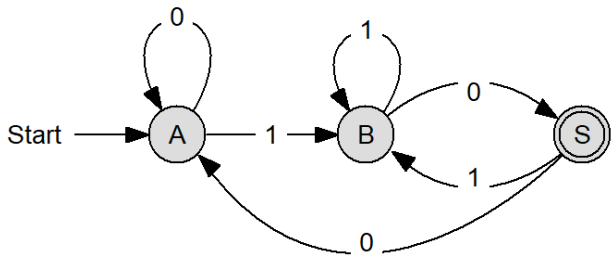

Abbildung 3.6: DEA für alle Worte, die auf 10 enden

Wir beginnen mit dem akzeptierenden Zustand und haben eine Kante, die mit 0 nach S führt und von B kommt. Nach der 1. Umsetzungsregel können wir die Regel aufstellen $S \to B0$. Nun betrachten wir den Zustand B. Dort gibt es eine Kante von A mit 1. Am Graphen können wir aber nicht sehen, ob diese 1 die erste im Wort ist, also direkt den Start des Wortes darstellt (dann müssten wir eine Regel $B \to 1$ aufstellen), oder ob es schon eine Vorgeschichte gibt, z.B. 0, so dass die 1 aus dem Zustand A kommt. Dann hieße die Regel $B \to A1$. Dieses Problem tritt immer dann auf, wenn der Startzustand nicht nur Durchgangsstation für die ersten Symbole, sondern selbst Ziel anderer Kanten ist.

Ist der Startzustand Zielpunkt anderer Kanten, dann teilen wir den Startzustand auf in einen reinen Durchgangsknoten und einen Knoten, der Ziel anderer Kanten ist.

Ein Beispiel für diese Aufteilung zeigt das veränderte Zustandsdiagramm für den Automaten, der alle Worte akzeptiert, die auf 10 enden:

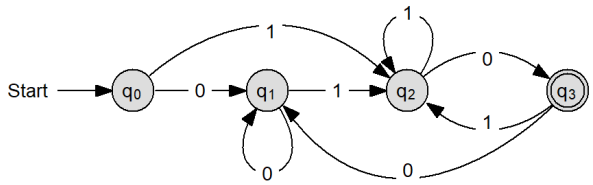

Abbildung 3.7: DEA für alle Worte, die auf 10 enden, mit Buchstaben

Wir ersetzen wiederum - mit Ausnahme des Startzustandes - die Bezeichnungen für die Zustände durch Großbuchstaben:

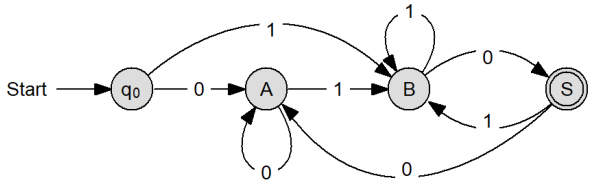

Abbildung 3.8: DEA für alle Worte, die auf 10 enden

Wir beginnen mit dem Startsymbol, d.h. dem akzeptierenden Zustand des DEA, und formulieren die Regeln der Grammatik umgekehrt wie die Richtung der Kanten. Zur Umsetzung benötigen wir zunächst Regeln der Form 3. Von Zustand B führt die Kante 0 in den akzeptierenden Zustand. Also ist unsere erste Regel $S \to B0$.

Dann betrachten wir den Zustand B. Wir haben eine Kante mit 1, die von B nach B geht. Die zugehörige Regel ist $B \to B1$. Eine weitere Kante mit 1 kommt vom akzeptierenden Zustand: $B \to S1$. Eine dritte Kante mit 1 kommt von A. Die Regel lautet $B \to A1$. Die letzte Kante mit 1 stammt schließlich vom Startzustand. Wir benötigen also eine Regel der Form 9.2. Die Regel lautet $B \to 1$.

Schließlich fehlt noch der Zustand A. Hier gibt es eine Kante mit 0 von A: $A \to A0$. Eine weitere Kante mit 0 stammt von S: $A \to S0$. Die letzte Kante mit 0 kommt vom Startzustand: $A \to 0$.

Damit können wir die Grammatik zusammenstellen:

Linksreguläre Grammatik G für Worte, die auf „10“ enden:

- V = {A, B, S}
- $\Sigma = \{0, 1\}$
- P = $\{S \to B0, A \to 0|A0|S0, B \to 1|B1|A1|S1\}$
- S Startvariable

3.4.2 DEA in rechtsreguläre Grammatik

Was ändert sich, wenn wir einen DEA in eine rechtsreguläre Grammatik umsetzen wollen? Eine rechtsreguläre Grammatik besteht (neben der terminierenden Regel Zustand → Eingabezeichen) aus Regeln der Form A → aB, in Worten: Ein Zustand wird ersetzt durch das Eingabezeichen und den Folgezustand.

Akzeptierende Zustände, die nur Kanten aufnehmen, können weggelassen werden. Wenn ein akzeptierender Zustand auch Kanten weiterführt, gibt es jedes Mal, wenn der Zustand erreicht wird, zwei Möglichkeiten: das Wort endet hier oder das Wort geht noch weiter. Entsprechend müssen für jede Kante, die in einen akzeptierenden Zustand hineinführt, zwei Grammatikregeln aufgestellt werden: eine terminierende Regel und eine rechtsregulär weiterführende Regel.

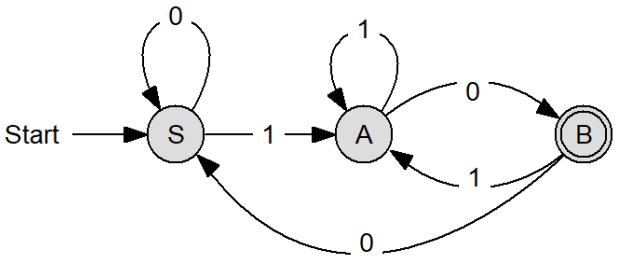

Abbildung 3.9: DEA für alle Worte, die auf 10 enden

Wir haben die Zustände bereits umbenannt. Der Startzustand wird in der Grammatik unser Startsymbol S. Da der akzeptierende Zustand über weiterführende Kanten verfügt, wird er mit B bezeichnet.

Wir stellen zunächst die Ersetzungsregeln für den Startzustand auf. Mit einer 0 landet man wieder bei S, also gilt S → 0S. Mit einer 1 landet man bei A, also gilt S → 1A. Von A führen zwei Kanten ab: Mit einer 1 landet man wieder bei A, also gilt: A → 1A. Mit einer 0 landet man im akzeptierenden Zustand B, dafür benötigen wir zwei Regeln: A → 0 und A → 0B. Von B landet man mit einer 0 in S, also B → 0S. Mit einer 1 gelangt man nach A, also B → 1A.

Damit lautet die rechtsreguläre Grammatik G für Worte, die auf „10“ enden:

- V = {A, B, S}
- Σ = {0, 1}
- P = $\{S \to 0S|1A, A \to 1A|0B|0, B \to 1A|0S\}$
- S Startvariable

3.4.3 Umsetzung einer linksregulären Grammatik in einen DEA

Ein vereinfachter Pascal-Bezeichner beginnt mit einem Buchstaben. Darauf folgen weitere Buchstaben oder Zahlen. Sonderzeichen sind nicht zugelassen. Um auch falsche Eingaben zu ermöglichen, wählen wir als Terminalsymbole neben „a“ (stellvertretend für Buchstaben) und „1“ (stellvertretend für Ziffern) auch „+“ für Sonderzeichen: $\Sigma = \{a, 1, +\}$.

Zur Vereinfachung schreiben wir Regeln mit der gleichen linken Seite rechts mit dem senkrechten Strich: $A \to Ba$ und $A \to Bb$ wird abgekürzt als $A \to Ba|Bb$.

- Jeder Pascal-Bezeichner beginnt mit a und a ist ein zulässiger Bezeichner: $S \to a$.
- Hinten dürfen Buchstaben folgen: $S \to Sa$.
- Hinten dürfen Zahlen folgen: $S \to S1$
- Zu Beginn sind weder Zahlen noch Sonderzeichen erlaubt: $F \to 1|+$
- Sonderzeichen sind in Bezeichnern nicht erlaubt: $F \to S+$
- Fehler sind nicht behebbar: $F \to Fa|F1|F+$

Das Regelsystem der Grammatik lautet also P = $\{S \to a, S \to Sa|S1, F \to +|1|S+|Fa|F1|F+\}$

Regel 1: $S \to a$ wegen $A \to a$:

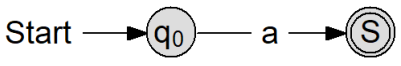

Abbildung 3.10: $S \to a$

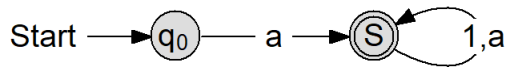

Abbildung 3.11: $S \rightarrow Sa|S1$

Regel 2: $S \rightarrow Sa|S1$ wegen $A \rightarrow Ba$:
Regel 3; $F \rightarrow +|1|S+|Fa|F1|F+$ wegen $A \rightarrow Ba$ und $A \rightarrow a$:

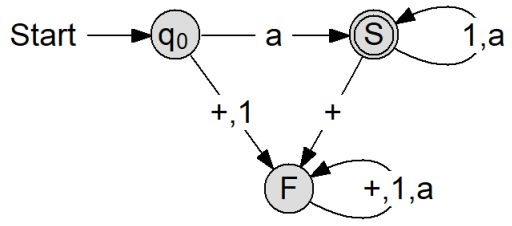

Abbildung 3.12: $F \rightarrow +|1|S+|Fa|F1|F+$

Damit ist die Grammatik in das Zustandsdiagramm eines deterministischen endlichen Automaten (DEA) umgesetzt.

Soweit das Beispiel aus einem in Niedersachsen weit verbreiteten Lehrwerk.[2] Allerdings haben wir damit eine ganze Reihe von völig überflüssigen Grammatikregeln gewonnen, nämlich alle, mit denen F ersetzt wird. Da eine Grammatik dazu dient, Worte der zugehörigen Sprache zu generieren oder zu überprüfen, vereinbaren wir, dass wir die Kanten, die in einen absorbierenden Fehlerzustand („Trap“) hineinführen oder von der Trap abgehen, bei einer Grammatik nicht berücksichtigen. Damit hat die Grammatik zu Abbildung 10.9. die Form $G = \{V = \{S\}, \Sigma = \{a, 1\}, P = \{S \rightarrow a|Sa|S1\}, S\}$.

3.4.4 Umsetzung einer regulären Sprache in eine Grammatik

Eine mögliche Aufgabenstellung ist die Umsetzung einer durch eine Aufzählung gegebene regulären Sprache in eine Grammatik. Sei beispielsweise L eine Sprache, die aus den Worten `1nnnnt, 1n2n1nt, 1nn2nt, 2nnt, 2n1nnt` und `4n` sowie beliebigen Aneinanderreihungen dieser sechs Worte besteht. Die direkte Aufstellung einer Grammatik, die in diesem Fall etwa 20 Ersetzungsregeln umfasst, ist auch für sehr gute Schülerinnen und Schüler kaum zu bewerkstelligen. Korrekturen am eigenen Produkt sind noch schwieriger, weil eine Menge von Ersetzungsregeln ab einer bestimmten Anzahl unübersichtlich wird. Solche Aufgaben zählen deshalb zum Anforderungsbereich III. Dagegen wird die Lösung geradezu ein Kinderspiel, wenn wir uns daran erinnern, dass endliche Automaten und reguläre Grammatiken äquivalent sind. Wir können also den Zustandsgraph des endlichen Automaten zeichnen, dort unsere Korrekturen vornehmen, bis die Sprache vollständig erkannt wird, und dann die Regeln der Grammatik an den Kanten des Graphen ablesen und notieren.

Betrachten wir die o.a. Sprache genauer. Drei Worte beginnen mit `1n`, zwei mit `2n`. Alle sechs Worte enden auf `nt`. Nach dem `t` sind wir wieder im Startzustand, weil ein beliebiges Wort der Sprache folgen kann. Daraus ergibt sich bereits ein Gerüst für unseren Automaten, wobei wir für die Zustände gleich Großbuchstaben verwenden, um die Umsetzung nachher zu erleichtern:

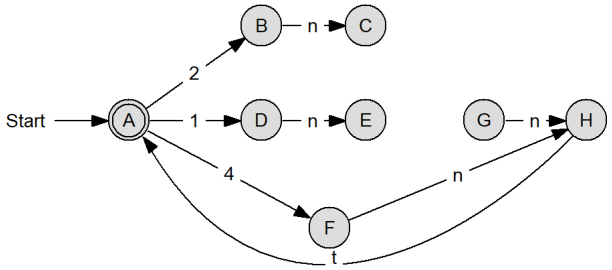

Abbildung 3.13: Grundgerüst des DEA für L

In den rechten Zustand H darf man nur mit `n` gelangen, weil alle Worte auf `nt` enden. Nun können wir die fehlenden Zustände und Übergänge ergänzen. Im Zustand C haben wir bereits die Eingabe von `2n` gehabt, wir können also mit `nt` das nächste Wort abschließen oder wir fügen ein `1n` ein und gehen dann mit `n` zu H. Von E kommen wir mit `nn` oder mit `n2` nach G oder mit `2n1n` direkt nach H.

An dem fertigen Graphen können wir testen, ob unsere sechs Worte akzeptiert werden oder ob noch zusätzliche Worte akzeptiert werden, die nicht zur Sprache gehören. Falls hier tatsächlich noch Fehler auftreten, lassen sie sich leicht eingrenzen und beheben. Im Ideal-

[2] E. Modrow, Theoretische Informatik mit Delphi, S. 95f

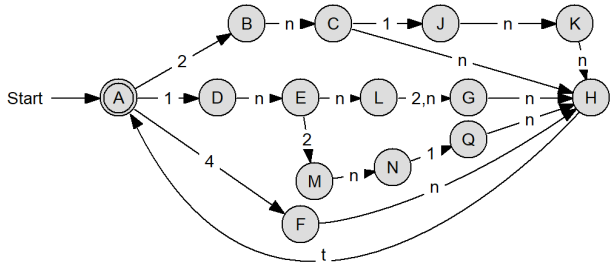

Abbildung 3.14: DEA für L

fall gibt es genau sechs Wege, die von A durch den Graphen wieder zu A führen, und diese entsprechen unseren sechs Worten.

Um die gesuchte Grammatik aufzustellen, gehen wir den Graphen in Richtung der Kanten durch und notieren die Regeln in der Form „Startzustand → Eingabezeichen Folgezustand“. Dabei gibt es eine Ausnahme: Jedes Mal, wenn wir in den akzeptierenden Zustand gelangen, kann das Wort zu Ende sein oder weitergehen. Wir benötigen also für jede Kante, die in den akzeptierenden Zustand führt, zwei Ersetzungsregeln, eine terminierende Regel der Form „A → a“ und eine fortführende Regel der Form „A → aB“.

Nach diesen Vorarbeiten können wir die gesuchte rechtsreguläre Grammatik notieren: $G = \{NT = \{A, B, C, D, E, F, G, H, J, K, L, M, N, Q\}; T = \{1, 2, 4, n, t\}; P : A \rightarrow 1D|2B|4F, B \rightarrow nC, C \rightarrow 1J|nH, D \rightarrow nE, E \rightarrow 2M|nL, F \rightarrow nH, G \rightarrow nH, H \rightarrow t|tA, J \rightarrow nK, K \rightarrow nH, L \rightarrow 2G|nG, M \rightarrow nN, N \rightarrow 1Q, Q \rightarrow nH; A\}$ Man beachte die doppelte Regel von H nach A, wovon eine terminierend ist.

3.4.5 Regularität von Sprache und Grammatik

Die Frage, ob eine Grammatik regulär ist und ob die zugehörige Sprache regulär ist, muss unterschieden werden.

Die Regularität einer Grammatik kann an der Form der Regeln festgemacht werden. Wenn alle Regeln der Form **A → Ba** und **A → a** sind, dann ist die Grammatik linksregulär. Wenn alle Regeln der Form **A → aB** und **A → a** sind, dann ist die Grammatik rechtsregulär. Man beachte, dass **jede Grammatik mindestens eine terminierende Regel A → a enthalten muss!** Sonst können keine Worte gebildet werden. Wenn eine Grammatik Regeln der Form A → aB **und** A → Ba enthält, ist sie nicht regulär. Eine reguläre Grammatik führt **immer** zu einer regulären Sprache.

Umgekehrt gilt das nicht. Wenn eine Grammatik nicht regulär ist, folgt daraus nicht automatisch, dass die zugehörige Sprache nicht regulär ist. Um die Regularität einer Sprache zu zeigen, ist - wie im Beispiel der Erstellung einer Grammatik - der „Umweg“ über den endlichen Automaten hilfreich. Wenn es einen deterministischen endlichen Automaten (DEA) gibt, der die Sprache erkennt, dann ist die Sprache regulär.

3.5 Aufgaben

Aufgabe 3.1 *Gegeben sei das folgende Zustandsdiagramm eines endlichen Automaten.*

Untersuchen Sie, ob die Worte 01010 und 101001 zu der durch das Zustandsdiagramm definierten Sprache gehören durch Angabe der durchlaufenen Zustände.

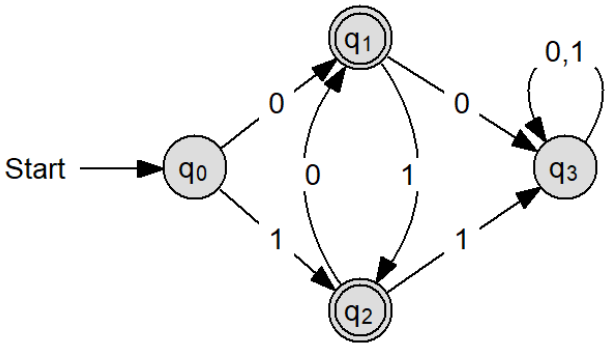

Abbildung 3.15: zu Aufgabe 10.1

Beschreiben Sie die durch den Automaten erkannte Sprache.

Stellen Sie eine rechtsreguläre Grammatik auf, die die Sprache erkennt.

Aufgabe 3.2 *Erstellen Sie das Zustandsdiagramm eines Automaten, an dem für 2 Euro in 1 Euro oder 2 Euro Stücken ein spezielles Buch ausgeliehen werden kann. Die Besonderheit dieses Automaten soll darin liegen, dass er nur maximal drei dieser Bücher beherbergen kann.*

Entsprechend soll er auf Eingaben reagieren, wenn er aktuell keine Bücher mehr hat bzw. zu viele Bücher versucht werden zurückzugeben.

Aufgabe 3.3 *Zeichnen Sie das Zustandsdiagramm eines endlichen Automaten mit dem Eingabealphabet $E = \{0, 1\}$, der die Worte erkennt, bei denen die Anzahl der Einsen durch 3 teilbar ist, also z.B. 000, 010110, 110110011.*

Geben Sie dazu eine linksreguläre Grammatik an.

4 Nichtdeterministische endliche Automaten (NEA)

In diesem Kapitel lernen Sie

- wie man mit nichtdeterministischen endlichen Automaten (NEA) Zustandsdiagramme konstruiert,
- wie man NEAs mit der vereinfachten Potenzmengenkonstruktion in DEAs überführt.

4.1 Funktionsweise und Definition des NEA

Die Forderung nach der Determiniertheit, also nach der Eindeutigkeit endlicher Automaten erscheint so einleuchtend, dass man sich fragt, warum man zusätzlich nichtdeterministische Automaten untersuchen soll. Auf diese Frage gibt es zwei Antworten:

Einerseits spielt der Begriff des Nichtdeterminismus eine große Rolle nicht nur in der Theoretischen Informatik, sondern auch als Lösungsstrategie. Es gibt viele Probleme, die streng deterministisch für große n praktisch nicht gelöst werden können, für die aber nichtdeterministische Lösungsverfahren existieren. Die nichtdeterministischen endlichen Automaten sind ein relativ einfaches Beispiel, um diesen Begriff einzufügen.

Andererseits können nichtdeterministische endliche Automaten (NEA) eine Hilfe für die Konstruktion von deterministischen Automaten sein. Bei vielen Aufgaben ist es einfacher, zunächst einen NEA zu konstruieren und diesen dann in einen DEA umzuwandeln. Wer den Algorithmus zur Umwandlung beherrscht, verfügt gewissermaßen über eine Geheimwaffe zur Lösung von DEA-Konstruktionsaufgaben.

Das erste Beispiel zeigt einen nichtdeterministischen endlichen Automaten (NEA), der

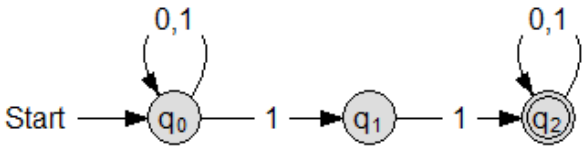

Abbildung 4.1: 1. Beispiel eines NEA

alle Worte akzeptiert, die die Folge 11 enthalten. Man erkennt im Zustand q0 den Nichtdeterminismus: es gehen zwei Kanten aus diesem Zustand ab, die mit beide mit „1“ beschriftet sind. Man erkennt den Nichtdeterminismus am einfachsten in der Tabelle der Überführungsfunktion δ, wo in einer Zelle mehr als ein Folgezustand auftaucht. Akzeptierende Zustände markieren wir durch ein nachgestelltes „e“:

NEA δ	0	1
q0	q0	q0, q1
q1	—	q2
q2 e	q2	q2

Das „Akzeptieren“ funktioniert bei den nichtdeterministischen endlichen Automaten anders als bei den deterministischen. Für jedes Eingabewort muss an jeder Stelle, an der der Automat nicht eindeutig ist, eine Fallunterscheidung durchgeführt werden. Wenn eine dieser Fälle in einen akzeptierenden Zustand führt, wird das Wort akzeptiert. Betrachten wir als Beispiel das Wort **10110**. Wir starten im Zustand q0. Im ersten Schritt lesen wir das Eingabezeichen 1 und haben zwei Möglichkeiten. Wenn wir nach q1 gehen, kommt als nächstes Eingabezeichen eine 0. Die ist aber im Zustand q1 nicht definiert. Wir müssen also hier abbrechen. Wenn wir mit dem ersten Buchstaben im Zustand q0 bleiben, folgt als zweites Eingabezeichen die 0.

Wir bleiben also im zweiten Schritt im Zustand q0. Als drittes Eingabezeichen erhalten

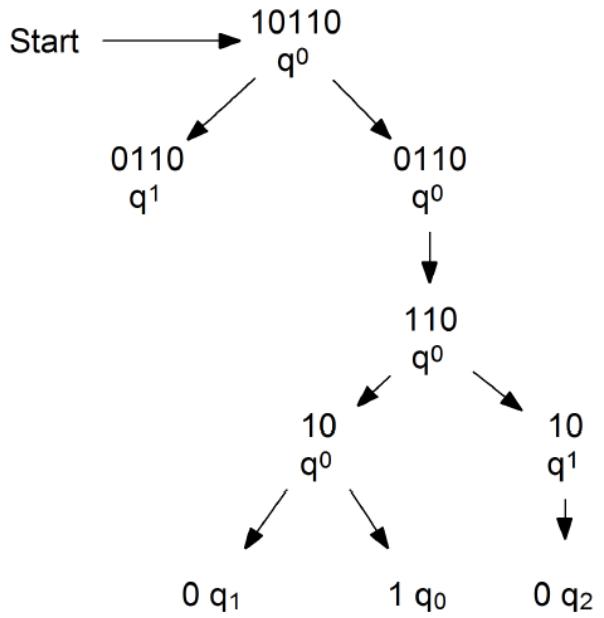

Abbildung 4.2: Entscheidungsbaum 10110

wir eine 1. Es gibt wieder zwei Möglichkeiten. Wenn wir mit dem vierten Zeichen nach q1 gehen, können wir die abschließende 0 nicht mehr umsetzen. Also bleiben wir mit dem vierten Zeichen im Zustand q0 und können dann das fünfte Zeichen zwar umsetzen, sind aber in q0 und damit in keinem akzeptierenden Zustand.

Wenn wir mit dem dritten Eingabezeichen (1) nach q1 gehen, müssen wir mit dem vierten nach q2. Mit der abschließenden 0 bleiben wir in q2. Wir haben also die gesamte Eingabefolge abgearbeitet und sind in einem akzeptierenden Zustand gelandet. Damit akzeptiert unser Automat das Wort **10110**.

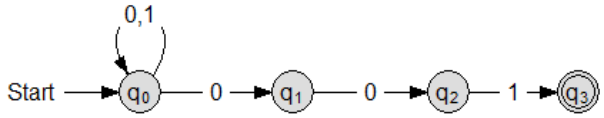

Abbildung 4.3: 2. Beispiel eines NEA

Das zweite Beispiel zeigt einen nichtdeterministischen endlichen Automaten (NEA), der alle Worte akzeptiert, die mit dem Teilwort „001“ enden.

Damit können wir den nichtdeterministischen Automaten definieren:

Definition:
Ein nichtdeterministischer endlicher Automat (NEA) besteht aus

NEA δ	0	1
q0	q0, q1	q0
q1	q2	—
q2	—	q3
q3 e	—	—

1. einer endlichen Menge von Zuständen Q,
2. einem Eingabealphabet Σ,
3. einer nicht eindeutigen Überführungsfunktion δ, die zu Zustand und Eingabezeichen eine Menge von Folgezuständen angibt (dabei ist δ in der Regel auch nicht vollständig),
4. einem Startzustand (hier q0),
5. einer endlichen Menge von akzeptierenden Zuständen E.

Der Nichtdeterminismus, also die Nichteindeutigkeit der Überführungsfunktion, charakterisiert den NEA. In der Regel verzichten wir auch auf die Forderung nach Vollständigkeit, um die Überführungsfunktion möglichst knapp zu lassen.

Es stellt sich nun die Frage nach dem Verhältnis der nichtdeterministischen und der deterministischen Automaten. Gibt es zu jedem nichtdeterministischen Automaten einen äquivalenten deterministischen Automaten oder ist die Menge der nichtdeterministischen Automaten mächtiger als die der deterministischen? Wir untersuchen diese Frage, indem wir ein Verfahren angeben, mit dessen Hilfe jeder nichtdeterministische endliche Automat in einen äquivalenten DEA umgewandelt werden kann.

4.2 Die Potenzmengenkonstruktion

Wie kann man nun einen nichtdeterministischen endlichen Automaten durch einen deterministischen endlichen Automaten ersetzen? Von der Definition her besteht der einzige Unterschied zwischen den beiden Typen in der Überführungsfunktion, weil in der Tabelle der Übergangsfunktion eines NEA nicht nur

einzelne Zustände, sondern Teilmengen von Zuständen auftauchen. Diese Beobachtung liefert den entscheidenden Lösungshinweis: Wenn wir statt der Zustände des NEA die Menge aller Teilmengen von Zuständen des NEA betrachten, ist die Überführungsfunktion wieder eindeutig.

Die Menge aller Teilmengen einer gegebenen Menge M bezeichnen wir als **Potenzmenge P(M)**. Daher stammt der Name für die Simulation eines NEA durch einen DEA.

Menge	Potenzmenge
∅ leere Menge	{∅}
{a}	{∅, {a}}
{a,b}	{∅, {a}, {b}, {a,b}}
{a,b,c}	{∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c},}

Allgemein gilt: Die Potenzmenge einer Menge M mit n Elementen besteht aus 2^n Elementen. Die Anzahl der Teilmengen mit jeweils k Elementen richtet sich nach den Zahlen aus dem Pascal'schen Dreieck bzw. nach den Binomialkoeffzienten. Die leere Menge hat nur sich selbst als Teilmenge. Die Menge mit einem Element hat zwei Teilmengen, die leere Menge und die Menge mit dem Element. Die Menge mit zwei Elementen hat vier Teilmengen, die leere Menge, zwei Teilmengen mit einem Element und eine Teilmenge mit zwei Elementen. Bei einer dreielementigen Menge gibt es eine Teilmenge mit drei Elementen (M selbst), drei Teilmengen mit zwei Elementen, drei Teilmengen mit einem Element und eine leere Teilmenge, also insgesamt acht Teilmengen.

Was passiert mit der Potenzmenge, wenn die ursprüngliche Menge ein zusätzliches Element erhält? Es gibt für jede der bisherigen Teilmengen zwei Möglichkeiten, entweder wird das neue Element hinzugefügt oder nicht. Das bedeutet, dass sich die Anzahl der Teilmengen immer verdoppelt, wenn sich die Anzahl der Elemente um Eins erhöht.

Wir betrachten nochmals die Überführungsfunktion des ersten Beispiel-NEAs:

Dabei war q2 der akzeptierende Zustand.

Um etwas Schreibarbeit zu sparen, schreiben

NEA δ	0	1
q0	q0	q0, q1
q1	—	q2
q2 e	q2	q2

wir die Teilmengen mit jeweils einem q und einem Index, also nicht {q0, q1, q2}, sondern {q012}. Damit besteht die Potenzmenge aus den Elementen {q012}, {q01}, {q02}, {q12}, {q0}, {q1}, {q2}, ∅ oder {}. Dabei gehören alle Teilmengen zu den akzeptierenden Zuständen, die q2 enthalten.

Die Zeilen für das neue δ des DEA erhalten wir, indem wir jeweils die entsprechenden Zeilen aus der Tabelle des NEA zusammenfassen. Die Zeilen für die einelementigen Teilmengen können wir aus der NEA-Tabelle direkt übernehmen. Die leere Menge führt bei jedem Eingabezeichen in die leere Menge. Als Ergebnis tritt die leere Menge nur dort auf, wo alle Teilmengen leer sind. So ergibt die Kombination von q0 und qleer q0. Akzeptierende Zustände sind alle, die eine 2 im Namen tragen, also q012, q02, q12 und q2.

DEA δ	0	1
∅	∅	∅
q0	q0	q01
q1	∅	q2
q2 e	q2	q2
q01	q0	q012
q02 e	q02	q012
q12 e	q2	q2
q012 e	q02	q012

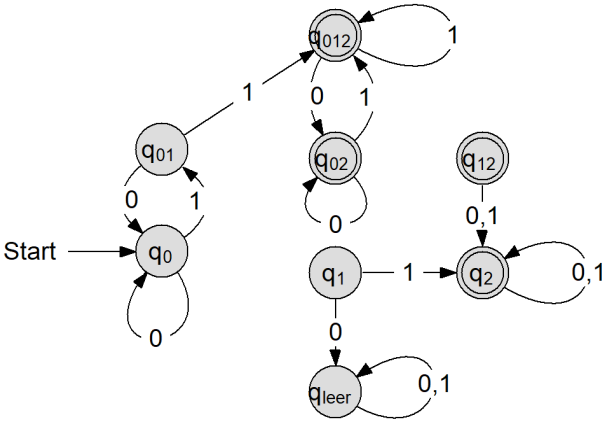

Abbildung 4.4: Potenzmenge des 1. Beispiels

Man erkennt, dass vier Zustände vom Startzustand aus gar nicht erreicht werden können, weil sie einen eigenen Teilgraphen bilden. Außerdem können im restlichen Graphen die beiden akzeptierenden Zustände zusammengefasst werden.

Die Überführungsfunktion für den zweiten Beispiel-NEA war, wobei wir den akzeptierenden Zustand durch ein nachgestelltes „e“ kennzeichnen:

NEA δ	0	1
q0	q01	q0
q1	q2	—
q2	—	q3
q3 e	—	—

Hieraus ergibt sich folgendes δ für die Potenzmenge:

DEA δ	0	1
$\emptyset$	$\emptyset$	$\emptyset$
q0	q01	q0
q1	q2	$\emptyset$
q2	$\emptyset$	q3
q3 e	$\emptyset$	$\emptyset$
q01	q012	q0
q02	q01	q03
q03 e	q01	q0
q12	q2	q3
q13 e	q2	$\emptyset$
q23 e	$\emptyset$	q3
q012	q012	q03
q013 e	q012	q0
q023 e	q01	q03
q123 e	q2	q3
q0123 e	q012	q03

Daraus ergibt sich der folgende Zustandsgraph:

Auch hier bildet die Hälfte der Zustände einen eigenen Teilgraphen, der vom Startzustand aus nicht erreicht werden kann. Beim genauen Hinsehen enthält der linke Teilgraph zudem vier Zustände, bei denen keine Übergänge ankommen, die also ebenfalls nicht erreicht werden können. Dies sind q02, q013, q023 und q0123. Damit reduziert sich der relevante Teil der Zustandsgraphen auf die vier Zustände q0, q01, q03 und q012.

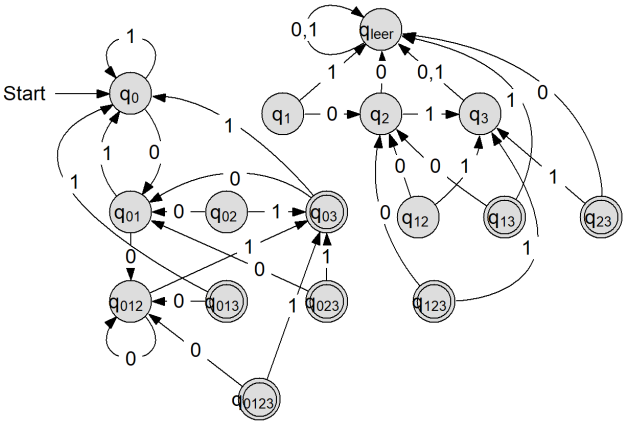

Abbildung 4.5: Potenzmenge des 2. Beispiels

Wir gehen noch einmal zurück zu unserer Ausgangsfrage: Ist die Menge der NEA genau so mächtig wie die Menge der DEA? Wir haben an dieser Stelle keinen formalen Beweis geführt, aber es ist mathematisch klar, dass wir zu jeder endlichen Menge mit n Elementen deren Potenzmenge mit 2^n Elementen bilden können. Wenn wir die Überführungsfunktion des NEA als Funktion der Potenzmenge beschreiben, dann erhalten wir einen eindeutigen und damit einen deterministischen endlichen Automaten, also einen DEA. Damit ist die Menge der NEA nicht mächtiger als die Menge der DEA.

Die konkreten Beobachtungen an den beiden Beispielen führen zu der Überlegung, bei der Umsetzung eines nichtdeterministischen endlichen Automaten (NEA) in einen deterministischen endlichen Automaten (DEA) **nur die relevanten Zustände der Potenzmenge** zu bestimmen. Dieser Gedanken führt uns zur vereinfachten Potenzmengenkonstruktion.

4.3 Die vereinfachte Potenzmengenkonstruktion

Alle relevanten Zustände werden vom Startzustand ausgehend irgendwann erreicht, d.h. sie tauchen in auf der rechten Seite der Tabelle auf. Wir beginnen wieder mit der Überführungsfunktion des 1. NEA:

Der Startzustand q0 ist immer erreichbar. Wir beginnen also mit der ersten Zeile. Auf der rechten Seite der Tabelle unterstreichen wir

NEA δ	0	1
q0	q0	q01
q1	—	q2
q2 e	q2	q2

alle neu auftretenden Kombinationen. Bei der handschriftlichen Bearbeitung kann man statt der Unterstriche auch einen kleinen Pfeil nach links unter den Zustand setzen, nach seinem Erfinder auch „Dükerpfeil“ genannt.

Für den DEA übernehmen wir zunächst die erste Zeile:

DEA δ	0	1
q0	q0	q01

Rechts taucht die neue Kombination q01 auf, die wir auf die linke Seite übernehmen. Für die rechte Seite fassen wir die Zeilen von q0 und q1 aus der NEA-Tabelle zusammen:

DEA δ	0	1
q0	q0	q01
q01	q0	q012

Rechts taucht die neue Kombination q012 auf, die wir wieder auf die linke Seite übernehmen. Da die Kombination den akzeptierenden Zustand q2 enthält, kennzeichnen wir sie mit „e“. Für die rechte Seite fassen wir die Zeilen von q0, q1 und q2 aus der NEA-Tabelle zusammen:

DEA δ	0	1
q0	q0	q01
q01	q0	q012
q012 e	q02	q012

Die neue Kombination q02 wird übernommen:

Es tauchen keine neuen Teilmengen der Potenzmenge mehr in der Tabelle auf, wir erhalten also genau den linken Teilgraphen von Beispiel 1, sogar mit den beiden akzeptierenden Zuständen:

DEA δ	0	1
q0	q0	q01
q01	q0	q012
q012 e	q02	q012
q02 e	q02	q012

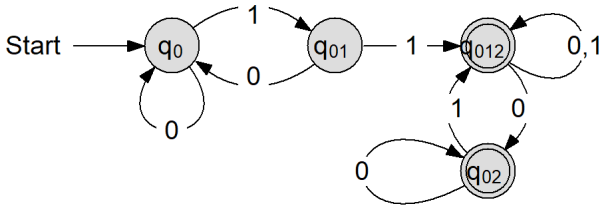

Abbildung 4.6: DEA für „enthält 11“

Wir können die beiden akzeptierenden Zustände zusammenfassen.

Entsprechend bearbeiten wir die Tabelle für das zweite Beispiel:

NEA δ	0	1
q0	q01	q0
q1	q2	—
q2	—	q3
q3 e	—	—

Wir erhalten für den zugehörigen DEA:

DEA δ	0	1
q0	q01	q0
q01	q012	q0
q012	q012	q03
q03 e	q01	q0

Damit ergeben sich ein Zustandsgraph aus den Zuständen, die wir bei der Potenzmengenkonstruktion als relevant bezeichnet hatten:

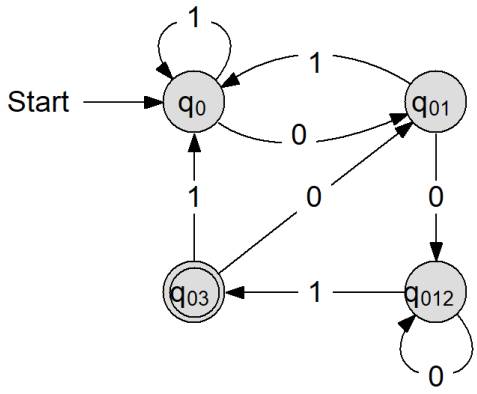

Abbildung 4.7: DEA für „endet auf 001“

Besonders einfache Grammatiken erhält man, wenn man auf den zugehörigen NEA ohne Trap zurückgreift.

4.4 Aufgaben

Aufgabe 4.1 *Gegeben ist die Überführungsfunktion eines NEA:*

NEA δ	A	C	G	T
q0	q1	-	-	-
q1	-	-	-	q2
q2	q1, q3	q3	q3	q3
q3	q3	q3, q4	q3	q3
q4	-	-	q5	-
q5	-	q4	-	-

Zeichnen Sie das Zustandsdiagramm des zugehörigen NEA (q_0 sei der Startzustand, q_5 der akzeptierende Zustand). Beschreiben Sie, welche Eingaben der Automat akzeptiert.

Überführen Sie mit Hilfe der vereinfachten Potenzmengenkonstruktion den Automaten in einen DEA und zeichnen Sie dessen Zustandsdiagramm.

Aufgabe 4.2 *Gegeben ist das folgende Zustandsdiagramm eines nichtdeterministischen endlichen Automaten.*

Überführen Sie mit Hilfe der vereinfachten Potenzmengenkonstruktion den Automaten in einen DEA und zeichnen Sie dessen Zustandsdiagramm.

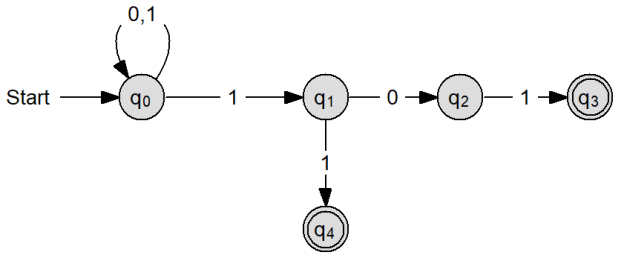

Abbildung 4.8: zu Aufgabe 4.2

5 Kellerautomaten und kontextfreie Grammatiken

In diesem Kapitel lernen Sie

- wie man kontextfreie Grammatiken für formale Sprachen entwirft,
- wie man eine reguläre Grammatik in einen endlichen Automaten überführt und umgekehrt,
- wie ein Kellerautomat aufgebaut ist und wie er funktioniert.

5.1 Grenzen endlicher Automaten

Eine wichtige Frage ist die Frage nach den Grenzen endlicher Automaten. Wir haben oben gesehen, dass die endlichen Automaten zu den regulären Grammatiken gehören, also zu den Typ-3-Grammatiken nach der Einteilung von Chomsky. Für welche Probleme reicht eine reguläre Grammatik nicht mehr aus? Wir betrachten dazu zunächst einige Beispiele.

Kann es einen deterministischen endlichen Automaten gibt, der alle Wörter akzeptiert, die gleich viele Zeichen a wie b beinhalten?

Eine solchen deterministischen endlichen Automaten, der genau die Wörter akzeptiert, die gleich viele Zeichen a wie b beinhalten, kann es nicht geben. Dieser hätte eine endliche Anzahl von Zuständen. Enthält das zu überprüfende Wort aber z.B. mehr Zeichen a hintereinander als der Automat Zustände hat, so kann der Automat die Zeichen a nicht korrekt abzählen.

Ein weiteres Beispiel sind geschachtelte Klammerstrukturen

- z.B. so: () (()) ()(()) ()()()
- oder so: ((())) ()() (()()(()()))

Ein Akzeptor für solche Strukturen muss sich „merken“ können, wie viele öffnende Klammern bisher aufgetreten und nicht durch schließende ausgeglichen worden sind. Dafür muss er zählen können!

Endliche Automaten können sich nur etwas merken, indem sie den Zustand wechseln. Hat ein solcher Automat n Zustände, dann gehen sie spätestens beim (n+1)-ten Zeichen in einen Zustand über, in dem sie schon vorher gewesen sind. Es ist danach nicht mehr feststellbar, auf welchem Weg sie diesen Zustand erreichten: sie haben den Zwischenweg „vergessen“. Ein endlicher Automat mit n Zuständen kann also nur von 0 bis (n-1) zählen.

Ähnliches gilt für die Palindrome. Ein Palindrom ist in der Informatik eine Zeichenkette, die vorwärts und rückwärts gelesen identisch ist, wie etwa „Reittier“ oder „Rentner“. Selbst wenn wir uns auf ein einfaches Alphabet wie die Dualzahlen beschränken, verdoppelt sich in jedem Schritt die Zahl der Möglichkeiten, die wir durch eigene Zustände abbilden müssten, um den Überblick haben. Auch hier reicht ein endlicher Automat nicht aus. Vielmehr müsste man in der ersten Hälfte des Wortes das entstehende Muster speichern, um dann in der zweiten Hälfte das gespeicherte Muster Zeichen für Zeichen mit dem eingegebenen zu vergleichen.

Wenn man über solche Plausibilitätsüberlegungen hinaus gehen will und einen echten Beweis dafür führen möchte, dass eine gegebene Sprache nicht durch einen regulären Automaten erkannt werden will, liefert das **Pumping Lemma** dafür eine Hilfestellung. Das Pumping Lemma geht von der Äquivalenz von regulären Sprachen, Grammatiken und endlichen Automaten aus.

Folgende Aussagen sind äquivalent:

- L ist eine reguläre Sprache.
- L wird durch eine reguläre Grammatik beschrieben.

- L wird durch einen deterministischen endlichen Automaten (DEA) erkannt.

Pumping Lemma für reguläre Sprachen

Für jede reguläre Sprache L gibt es eine natürliche Zahl **n**, sodass gilt: Jedes Wort z in L mit Mindestlänge **n** hat eine Zerlegung z=uvw mit den folgenden drei Eigenschaften:

1. Die beiden Wörter u und v haben zusammen höchstens die Länge n.
2. Das Wort v ist nicht leer.
3. Für jede natürliche Zahl i (einschließlich 0) ist das Wort uv^iw in der Sprache L, d. h. die Wörter uw, uvw, uvvw, uvvvw usw. sind alle in der Sprache L.

Das kleinste n, das diese Eigenschaften erfüllt, wird Pumping-Zahl der Sprache L genannt.

Die Gültigkeit des Lemmas basiert darauf, dass es zu jeder regulären Sprache einen deterministischen endlichen Automaten gibt, der die Sprache akzeptiert. Über einem endlichen Alphabet enthält eine reguläre Sprache mit unendlich vielen Wörtern auch solche Wörter, die mehr Zeichen enthalten als der Automat Zustände hat. Zum Akzeptieren solcher Wörter muss der Automat also einen Zyklus enthalten, der dann in beliebiger Häufigkeit durchlaufen werden kann. Die Buchstabenfolge, die beim Durchlaufen des Zyklus gelesen wird, kann also entsprechend beliebig oft in Wörtern der Sprache vorkommen.

Alle Worte, die länger sind als n, lassen sich zerlegen in einen Anfang (Präfix) u, einen Mittelteil v (der wiederholt wird) und ein Ende (Suffix) w. Der DEA kann nicht unterscheiden, wie oft der Mittelteil wiederholt wird. Alle Worte, die durch „Aufpumpen“ / Wiederholen des Mittelteils entstehen, gehören ebenfalls zur Sprache, deshalb der Name „Pumping Lemma“.

Die Idee des Pumping-Lemmas ist, dass ein Wortteil durch einen Zyklus im deterministischen endlichen Automaten beliebig oft wiederholt werden kann.

Wir betrachten die Sprache $L = \{a(bc)^nd | n \in \mathbb{N}\}$. Zu L gehören u.a. die Worte ad, abcd, abcbcd, usw. Alle Worte, die mindestens vier Buchstaben umfassen, lassen sich in einen Anfang (a), einen Mittelteil, der wiederholt wird (bc), und einen Endteil (d) zerlegen. Dann ist die Länge von Anfang und Mittelteil zusammen 3, der Mittelteil ist nicht leer und jedes Wort, das aus einer Vervielfachung des Mittelteils hervorgeht, gehört ebenfalls zur Sprache L.

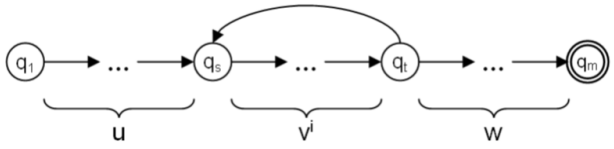

Abbildung 5.1: Veranschaulichung des Pumping Lemma

Ist die Sprache L=$\{a^mb^m, m \geq 1\}$, also L = $\{ab, aabb, aaabbb, aaaabbbb, usw.\}$ regulär?

Angenommen, L sei eine reguläre Sprache. Dann gibt es eine Zahl **n**, so dass sich alle Worte $z \in L$ mit $|z| = n$ wie beschrieben zerlegen lassen. Insbesondere gibt es eine Zerlegung uvw mit den beschriebenen Eigenschaften für das Wort $a^mb^m \in L$.

Da uv ein Präfix dieses Wortes ist und gemäß Eigenschaft 1 höchstens Länge n hat, besteht uv und damit v ausschließlich aus Buchstaben a.

Gemäß Eigenschaft 3 (für i=2) muss auch das Wort $w = a^{n+|v|}b^n$ in L liegen (Man müsste den Mittelteil wiederholen dürfen). Da aber $|v| > 0$ (Eigenschaft 2), enthält dieses Wort mehr a als b, liegt also nicht in L.

Also führt die Annahme, L sei eine reguläre Sprache, zum Widerspruch und ist damit falsch.

Anschaulich gesprochen: weil man jede Wiederholung mitzählen muss, um ein richtiges Wort zu konstruieren, ist die Sprache L=$\{a^mb^m, m \geq 1\}$ nicht regulär.

Das Pumping Lemma eignet sich nicht dazu, die Zugehörigkeit einer Sprache zur Klasse der regulären Sprachen nachzuweisen. Man kann damit nur zeigen, dass eine Sprache **nicht regulär** ist. Es gibt auch kontextfreie Sprachen, die die Bedingungen des Pumping Lemma erfüllen.

5.2 Kellerautomaten

Können in einer Grammatik Nichtterminalsymbole ohne Rücksicht auf den Kontext, in dem das zu ersetzende Symbol auftaucht, durch Symbolfolgen aus Terminal- und Nichtterminalsymbole ersetzt werden, dann spricht man von kontextfreien Grammatiken. Diese können von Kellerautomaten analysiert werden. Ein klassisches Beispiel von Kellerautomaten sind Automaten, die Klammerstrukturen auf ihre Richtigkeit untersuchen können. Bei Klammerstrukturen gilt, dass insgesamt die Anzahl der öffnenden und der schließenden Klammern gleich sein muss. Öffnende Klammern dürfen überall stehen. Für schließende Klammern gilt, dass an jeder Stelle des Eingabewortes die Anzahl der schließenden Klammern nicht größer sein darf, als die Anzahl der öffnenden Klammern, die bisher vorgekommen sind.

Um beliebige Klammerstrukturen analysieren zu können, muss der endliche Automat also um einen Speicher erweitert werden. Diesen organisieren wir in Form eines Stapel (Stack, Keller). Daher kommt der Name **Kellerautomat**.

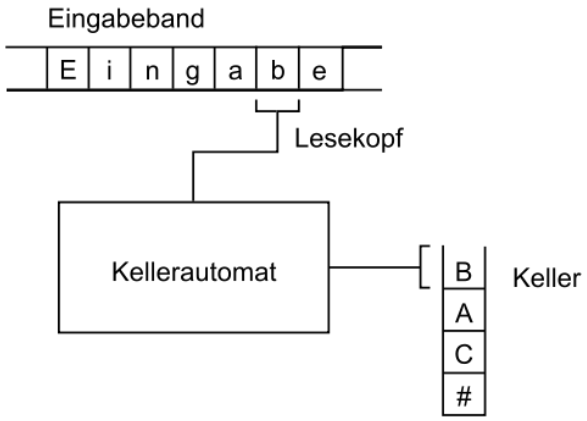

Abbildung 5.2: Funktionsweise eines Kellerautomaten

Definition eines Kellerautomaten

Ein nichtdeterministischer Kellerautomat (NKA) ist ein 7-Tupel (Z, A, K, d, S, #, E)

- Z ist die Zustandsmenge, eine nicht leere Menge von Zuständen.
- A ist das Eingabealphabet, eine endliche, nicht leere Menge von Symbolen.
- K ist das Kelleralphabet, eine endliche, nicht leere Menge von Symbolen. Wir erhöhen die Lesbarkeit, wenn wir für Eingabealphabet und Kelleralphabet unterschiedliche Buchstaben verwenden, z.B. Klein- und Großbuchstaben.
- d ist die Übergangsfunktion, die jeder Kombination aus Zustand, Kellersymbol und Eingabesymbol eine oder mehrere Kombination(en) aus Folgezustand und einer (ggf. leeren) endlichen Folge von Kellersymbolen zuordnet: $d : (z_i, k_j, a_l \rightarrow z_m,$ kfz, wobei kfz eine beliebige (auch leere) Folge von Kellerzeichen ist.
- S ist der Anfangszustand; S ist ein Element aus Z.
- # ist das Kellerstartsymbol; # ist ein Element aus K.
- E ist die Menge der akzeptierenden Zustände, eine Teilmenge von Z.

Wir konstruieren unsere Kellerautomaten so, dass in jedem Schritt das oberste Kellerzeichen gelesen und aus dem Stapel gelöscht wird (pop statt top). Benötigen wir das oberste Kellerzeichen in einem späteren Zustand, so muss es wieder in den Keller geschrieben werden.

Bei der Beschreibung der Übergangsfunktion beschränken wir uns auf die „richtigen“ Zustände. Ist für eine Kombination kein Übergang definiert, so bleibt der Kellerautomat stehen. Die Notation der Übergänge erfolgt etwas anders als beim endlichen Automaten. Die Übergangsfunktion eines Kellerautomaten ordnet einer Kombination aus Zustand, oberstem Kellersymbol und Eingabezeichen einen Folgezustand und eine beliebige Folge von Kellerzeichen zu. Innerhalb eines Zustandsdiagramms sind Zustand und Folgezustand ja durch Anfang und Ende der Kante gegeben. Wir notieren also - in Anlehnung an die Schreibweise bei AutoEdit - (oberstes Kellerzeichen, Eingabezeichen) : Kellerzeichenfolge, also oberstes Kellerzeichen und Eingabezeichen in einer runden Klammer und nach dem Doppelpunkt die Kellerzeichenfolge, die abgespeichert wird.

Wenn wir die Zustandsfunktion als Tabelle schreiben, müssen wir zusätzlich die Zustände

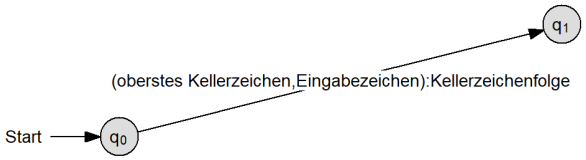

Abbildung 5.3: Übergang eines Kellerautomaten

angeben: (Zustand, oberstes Kellerzeichen, Eingabezeichen) : (Folgezustand, Kellerzeichenfolge).

Die Definition des Kellerautomaten als eines endlichen Automaten mit einem zusätzlichen Stack legt auch die Möglichkeiten fest, die ein Kellerautomat bietet. Der Stack wird von oben befüllt und wird auch von oben entleert. Wir beschränken uns dabei auf zwei Stack-Befehle: **push()** und **pop()**, wobei wir das pop in Verbindung mit dem Kellerstartzeichen dazu verwenden, einen leeren Stapel zu erkennen. Daraus ergeben sich drei Grundaufgaben für Kellerautomaten:

- Man kann ein Muster im Stack ablegen und beim Abbau des Stapels prüfen, ob das Muster in umgekehrter Reihenfolge wiederholt wird. Diese Aufgabe läuft auf das Erkennen von Palindromen hinaus, wobei es eine einfachere Variante mit einem festen Mittelzeichen gibt (dann reicht ein deterministischer Kellerautomat) oder ein echtes Palindrom (dann benötigt man einen nichtdeterministischen Kellerautomaten, weil wir nicht wissen, ab wann der gespiegelte Teil beginnt.

- Man kann einen Stapel aufbauen und beim Abbau des Stapels prüfen, ob ein zweiter Stapel genau so hoch ist. Man kann auch zwei Stapel nacheinander aufbauen und dann prüfen, ob die nächsten beiden Stapel gleich hoch sind, wobei die Reihenfolge vertauscht wird. Man kann aber nicht prüfen, ob drei Stapel gleich hoch sind. Die Information über die Höhe eines Stapels steckt im Stapel selbst und ist nach dessen Abbau verloren.

- Man kann prüfen, welches von zwei Eingabezeichen häufiger vorkommt. Dazu verwenden wir einen kleinen Trick. Beim ersten Auftreten eines Eingabezeichens gehen wir in den Zustand „mehr von...“, aber kellern das Eingabezeichen noch nicht ab. Erst beim zweiten Auftreten kellern wir das Zeichen ab. Tritt das andere Zeichen auf, löschen wir das oberste Kellerzeichen. Damit können wir erkennen, wenn der Zustand „gleich viel von beiden“ erreicht wird, nämlich wenn wir ein Kellerstartzeichen erhalten.

Ein Kellerautomat terminiert, d.h. beendet, wenn er sich nach Abarbeitung des Eingabewortes **in einem akzeptierenden Zustand** befindet **und** das **Kellerband** dann **leer ist**. **Achtung:** In AutoEdit reicht es, wenn sich der Automat in einem akzeptierenden Zustand befindet, auch wenn der Keller nicht leer ist. Beide Möglichkeiten sind äquivalent, führen aber auf unterschiedliche Kellerautomaten.

5.3 Kontextfreie Grammatiken

Eine Grammatik heißt kontextfrei, wenn gilt:

Jedes Nichtterminalsymbol wird ersetzt durch eine beliebige Folge von Terminalsymbolen und Nichtterminalsymbolen, wobei der Kontext (die Umgebung) keine Rolle spielt.

Alle Regeln sind von der Form $A \rightarrow \alpha$. Auch kontextfreie Grammatiken benötigen mindestens eine terminierende Regel der Form $A \rightarrow a$, d.h. ein Nichtterminalsymbol wird ersetzt durch ein Terminalsymbol (oder das leere Zeichen ϵ).

Beispiel: Die Grammatik G mit $G = \{\{S\}, \{0,1\}, S \rightarrow 0S0|1S1|\epsilon, S\}$ ist kontextfrei. Die Kontextfreiheit lässt sich übrigens nicht an der rechten Seite der Regeln erkennen, wo durchaus Terminalzeichen und Nichtterminalzeichen in unterschiedlichen Kombinationen auftreten können, sondern an der linken Seite, wo immer nur genau ein Nichtterminalzeichen stehen darf.

Kontextfreie Sprachen werden sowohl mit kontextfreien Grammatiken wie auch mit nichtdeterministischen Kellerautomaten vollständig beschrieben.

Deterministische Kellerautomaten beschreiben nur eine Teilmenge, die deterministisch kontextfreien Sprachen.

Für die kontextfreien Grammatiken gilt - wie bereits für die regulären Grammatiken -: die Lesbarkeit steigt, wenn wir die Terminalsymbole mit kleinen Buchstaben (bzw. mit Zahlen) und die Nichtterminalsymbole mit Großbuchstaben schreiben.

5.4 Syntaxdiagramme

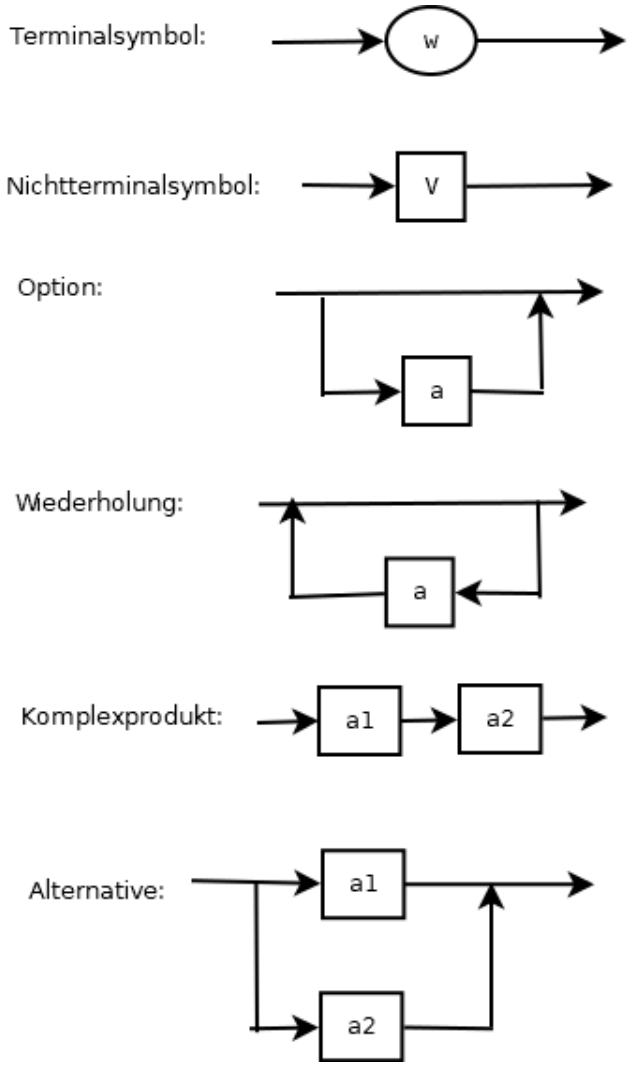

Abbildung 5.4: Elemente von Syntaxdiagrammen

Eine äquivalente Beschreibung von kontextfreien Grammatiken liefern Syntaxdiagramme. Wir notieren Nichtterminalsymbole bzw. Variable in rechteckigen Kästchen, Terminalsymbole bzw. Buchstaben in runden bzw. ovalen Kästchen.

Die Pfeile drücken Verzweigung (hier: Alternative), Wiederholung und Sequenz (hier: Komplexprodukt) aus. Hinzu kommt die Option.

5.5 Beispiele für Kellerautomaten

Sei L_1 die Sprache aller Worte über dem Alphabet $A = \{a, b\}$, die gleich viele a's wie b's enthalten, formal: $L_1 = \{\{a, b\}^* || w|_a = |w|_b\}$.

Wir benötigen drei Zustände: $a > b$ steht für „mehr a als b". $b > a$ steht für „mehr b als a"und $a = b$ steht für den akzeptierenden Zustand gleichviel a's wie b's. Die Anzahl der überzähligen a's zählen wir, indem wir A's auf den Stapel legen. Die Anzahl der überzähligen b's zählen wir, indem wir B's auf den Stapel legen. Dabei tritt das Problem auf, dass wir erkennen müssen, wann gleichviel a's und b's in der bisherigen Eingabe auftreten. Nehmen wir ein Beispiel: Wir haben zu Beginn des Wortes drei a's und legen dafür drei A's auf den Stapel. Wenn wir jetzt zwei b's bekommen, nehmen wir zwei A's vom Stapel. Wenn wir das dritte b bekommen, ist immer noch ein A oben auf dem Stapel. Wir erkennen also nicht, dass die Anzahl des a's und b's bereits ausgeglichen ist. Erst im nächsten Schritt erkennen wir, dass der Stapel leer ist. Dann haben wir aber bereits das 7. Zeichen. Die Lösung besteht darin, dass wir beim ersten a bzw. beim ersten b in in entsprechenden Zustand ($a > b$ bzw. $b > a$ gehen, aber noch kein A bzw. B auf den Stapel legen, sondern nur das Kellerstartzeichen. Wenn dann der Stapel abgebaut ist, erkennen wir beim letzten b oder a, dass er leer ist und dass nun einschließlich des aktuellen Eingabezeichens Gleichstand ist.

Im Zustand $a > b$ legen wir für jedes a, dass wir einlesen, ein weiteres A auf den Stapel. Für jedes b löschen wir ein A. Wenn wir das Eingabezeichen b lesen und der Stapel leer ist, gehen wir in den Zustand a=b. Entsprechendes gilt für den Zustand $b > a$. Im Zustand a=b gehen wir in den Zustand, dessen Mehrheit dem Eingabezeichen entspricht, speichern aber wie am Anfang nur das Kellerstartzeichen ab.

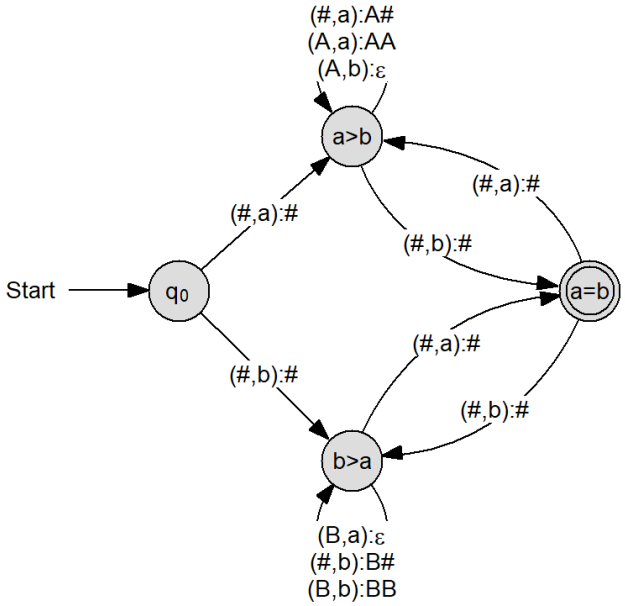

Abbildung 5.5: Kellerautomat zu L_1

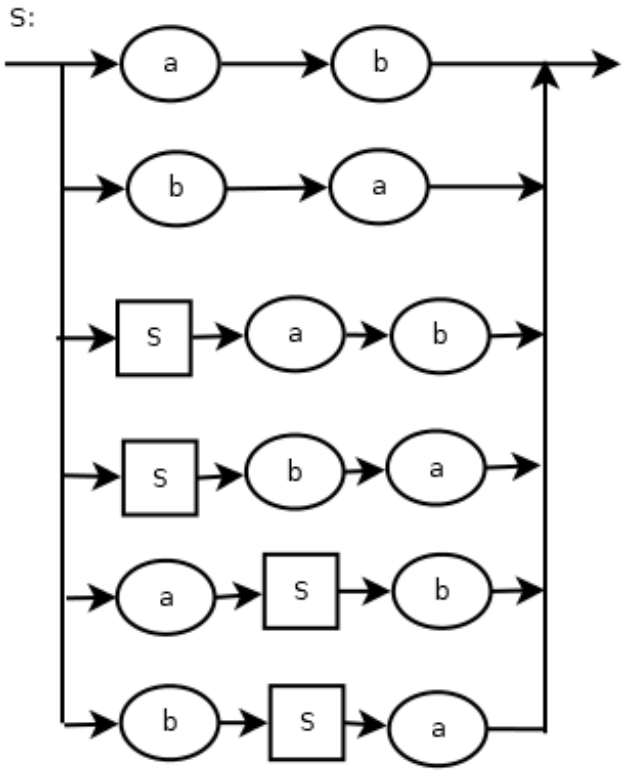

Abbildung 5.6: Syntaxdiagramm von L_1

Die zugehörige Grammatik erhalten wir nicht wie bei endlichen Automaten durch ein einfaches Umsetzungsverfahren. Die Zustände im Kellerautomaten entsprechen nicht den Nichtterminalzeichen in der kontextfreien Grammatik. Wir können die Worte der Sprache L_1 konstruieren, indem wir in jedem Schritt ein a und ein b anhängen. Wenn wir das leere Wort ausschließen möchten, können wir im ersten Schritt das Startsymbol durch ab oder durch ba ersetzen. Das wären die beiden kürzesten Worte der Sprache. Wollen wir das leere Wort einschließen, dann können wir im ersten Schritt das Startsymbol durch Epsilon ersetzen. Im nächsten Schritt ergänzen wir die Regeln durch die Möglichkeit, vor dem Startsymbol ein a und hinter dem Startsymbol ein b anzufügen bzw. umgekehrt. Wenn man testet, ob mit diesen Regeln ($S \rightarrow ab|ba|aSb|bSa$) alle vierbuchstabigen Worte der Sprache erkannt werden, stellt man fest, dass z.B. die Worte abba und baab damit nicht abgeleitet werden können. Man muss die beiden Buchstaben auch auf beiden Seiten des Startsymbols anfügen können. Damit ergibt sich folgende Grammatik: $G(L_1) = \{\{S\}, \{a, b\}, S \rightarrow ab|ba|aSb|bSa|Sab|Sba|abS|baS, S\}$

Da das Regelsystem der Grammatik aus einer einzigen Regel besteht, besteht auch das Syntaxdiagramm aus einem Diagramm:

Unser zweites Beispiel ist die Sprache der Klammerausdrücke L_2. Für diese Sprache gelten eine Reihe von Regeln:

- Zu Beginn steht immer eine öffnende Klammer.
- Öffnende Klammern dürfen an jeder Stelle stehen.
- Insgesamt ist die Anzahl der schließenden Klammern gleich der Anzahl der öffnenden Klammern.
- Zu jedem Zeitpunkt darf es höchstens so viele schließende Klammern geben, wie vorher öffnende Klammern vorgekommen sind.

Wenn wir einen leeren Stapel haben, kann nach den Regeln nur eine öffnende Klammer Eingabezeichen sein. In diesem Fall legen wir ein K auf dem Stapel ab. Haben wir das Eingabezeichen öffnende Klammer und K als oberstes Kellerzeichen, dann legen wir ein weiteres K auf dem Stapel ab. Für jede schließende Klammer entfernen wir ein K aus dem Stapel. Wenn die Zahl der öffnenden Klammern gleich der Zahl der schließenden Klammern war, ist am Ende der Eingabefolge der Stapel leer.

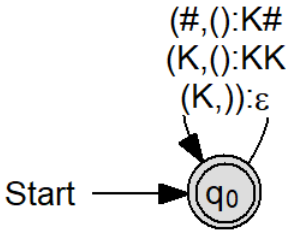

Abbildung 5.7: Kellerautomat der Klammerausdrücke

Für die zugehörige Grammatik überlegen wir: Zwei Klammernpaare können nebeneinander stehen. Ein Klammernpaar kann weitere enthalten oder kein weiteres enthalten. Daraus ergeben sich drei Ersetzungsregeln $S \rightarrow SS|(S)|()$. Die Grammatik ist somit $G(L_2) = \{\{S\}, \{(,)\}, S \rightarrow SS|(S)|(), S\}$.

Da auch hier das Regelsystem aus einer einzigen Regel besteht, ist das Syntaxdiagramm sehr einfach:

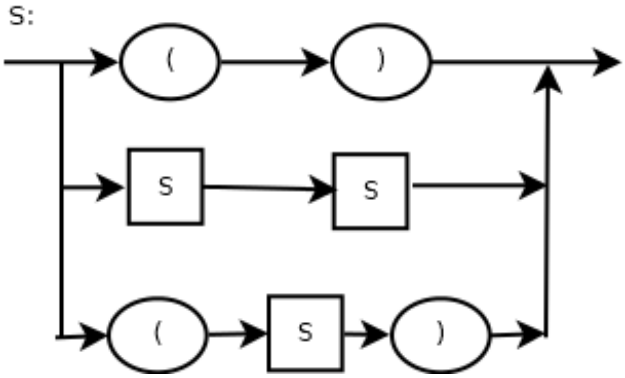

Abbildung 5.8: Syntaxdiagramm von L_2

Als drittes Beispiel betrachten wir nun die Sprache der Palindrome L_3 über der Menge $\{a, b, m\}^*$ mit mindestens drei Buchstaben. Dabei beschränken wir uns zunächst solche Palindrome, die einen festgelegten Buchstaben in der Mitte haben (hier m), der nur an dieser Stelle vorkommt. Zu dieser Sprache gehören also z.B. die Worte ama, bmb, abmba, aamaa, bamab und bbmbb. Für den zugehörigen Kellerautomaten benötigen wir zwei Zustände: Im ersten Zustand bauen wir den Stapel auf, indem wir entsprechend der Eingabezeichen mit A's und B's füllen. Dabei gibt es sechs Möglichkeiten: Der Stapel ist noch leer, das letzte Zeichen war ein a und das letzte Zeichen war ein b, jeweils kombiniert mit den Eingabezeichen a und b. Im zweiten Zustand bauen wir den Stapel ab, bis er leer ist. Hier müssen Eingabezeichen und Kellerzeichen übereinstimmen, sonst ist das Wort kein Palindrom. Der Übergang von einem zum anderen Zustand erfolgt durch den Buchstaben m, wobei wir das alte Kellerzeichen wieder sichern müssen.

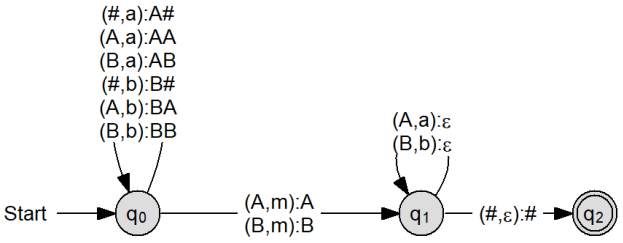

Abbildung 5.9: Kellerautomat zu L_3

Bei der Grammatik achten wir darauf, dass das Wort m nicht zu L_3 gehört: $G(L_3) = \{\{a, b, m\}, \{S, A\}, S \rightarrow aAa|bAb, A \rightarrow aAa|bAb|m, S\}$.

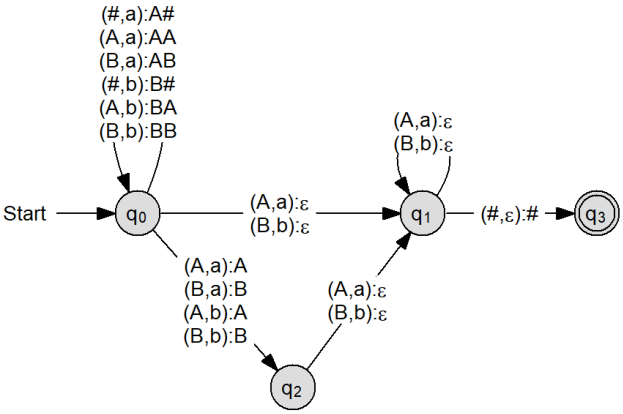

Abbildung 5.10: Kellerautomat echter Palindrome

Als viertes Beispiel betrachten wir die Sprache der echten Palindrome L_4 über der Menge $\{a, b\}^*$ mit mindestens zwei Buchstaben. Im Unterschied zu den vorigen Beispielen können wir jetzt keinen deterministischen Kellerautomaten verwenden. Bei einem echten Palindrom ist beim zeichenweisen Durchgehen nicht ersichtlich, wo der „Umkipp-Punkt“ ist, also wo die bisher eingegebenen Zeichen in umgekehrter Reihenfolge auftauchen. Der Automat muss also die verschiedenen Möglichkeiten durchprobieren. Das leistet ein nichtdeterministischer Kellerautomat (NKA). Außerdem gibt es einen weiteren Unterschied zu L_3. Das Palindrom

kann eine gerade Anzahl von Zeichen haben, dann wird der Stapel zunächst aufgebaut und dann wieder abgebaut. Das Palindrom kann aber auch eine ungerade Anzahl von Zeichen haben. Das Zeichen in der Mitte steht dann quasi für sich allein, es darf nicht abgespeichert werden, aber es darf auch kein Zeichen aus dem Stapel löschen. Damit gibt es für diesen Fall einen weiteren Zustand und wir erhalten den Zustandsgraphen.

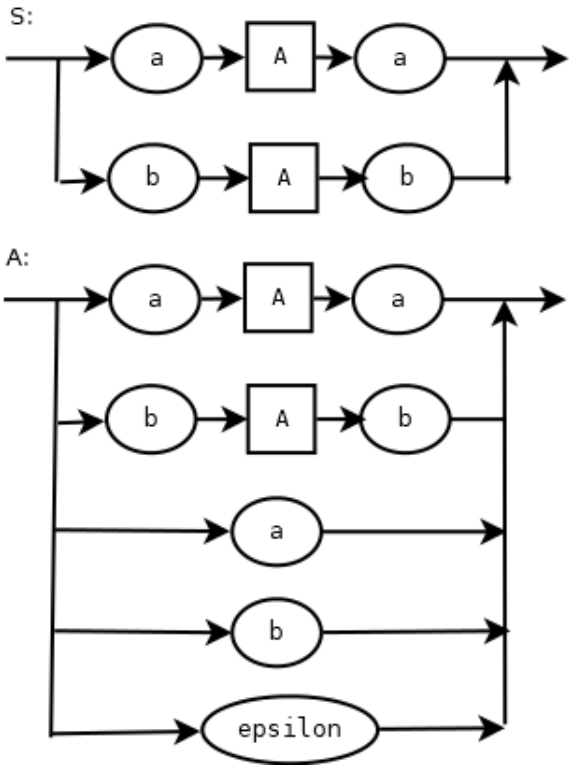

Abbildung 5.11: Syntaxdiagramm echter Palindrome

Bei der Grammatik kommen wir mit zwei Nichtterminalsymbolen aus: $G(L_4) = \{\{a, b\}, \{S, A\}, S \rightarrow aAa|bAb, A \rightarrow aAa|bAb|a|b|\epsilon, S\}$.

5.6 UML-Diagramm und Implementierung

Gegeben sei der folgende Kellerautomat:

Gesucht sind das zugehörige UML-Diagramm und eine Implementierung in Snap!.

Für das UML-Diagramm muss man sich zunächst überlegen, über welche weiteren Klas-

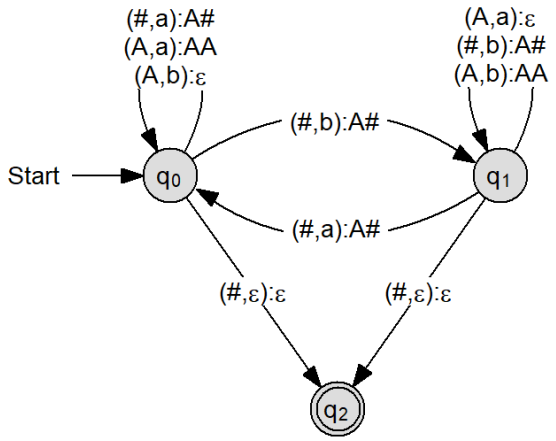

Abbildung 5.12: Beispiel Kellerautomat

sen der Kellerautomat verfügt. Er benötigt natürlich einen Keller. Für den Keller ist der naheliegende Datentyp ein Stapel, weil auf den Keller nach dem LIFO-Prinzip (last in, first out) geschrieben wird. Außerdem haben wir eine Reihe von Listen bzw. verketteten Listen in der Definition des Kellerautomaten. Wir benötigen also Aggregationen zu Stapel und Liste. Die Attribute des Kellerautomaten ergeben sich zunächst aus dessen Definition: Wir benötigen eine Liste von Zuständen, ein Eingabealphabet, ein Stackalphabet (beides ebenfalls Listen), eine Überführungsfunktion, die wir als verkettete Liste implementieren, einen Startzustand, eine Stackvorbelegungszeichen und eine Liste von akzeptierenden Zuständen. Diese sieben Attribute ergeben sich aus der Definition. Zusätzlich benötigen wir einen Keller und ein Eingabeband, das wir als Zeichenkette implementieren.

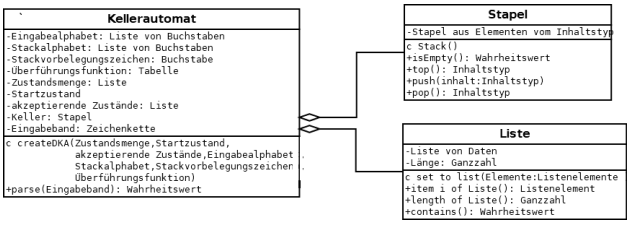

Abbildung 5.13: Klassendiagramm des Kellerautomaten

Der Kellerautomat kommt mit zwei Methoden aus: Er benötigt wie alle konkreten Klassen einen Konstruktor `createDKA`, der die sieben Bestandteile als Parameter erhält. Eine Methode `parse` gibt an, ob eine Eingabefolge zur der

vom Automaten akzeptierten Sprache gehört oder nicht.

Abbildung 5.14: Methode createDKA

Unsere `createDKA`-Methode ist nichts anderes als ein Reporter, der die sieben Bestandteile in eine Liste packt.

Den Parser zerlegen wir in zwei Blöcke, um ihn überschaubar zu halten. Ein Block `neuerZustand` schaut in der Tabelle der Überführungsfunktion nach, ob die aktuelle Kombination aus Zustand, Keller- und Eingabezeichen dort verzeichnet ist und liefert dann den neuen Zustand. Außerdem soll der Block gleich dafür sorgen, dass der Keller aktualisiert wird. Außer Zustand, Kellerzeichen, Eingabezeichen und Überführungsfunktion erhält er deshalb auch den Keller.

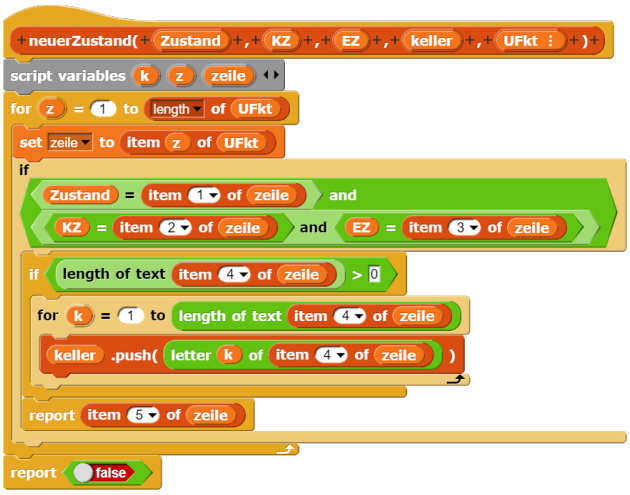

Abbildung 5.15: Block neuerZustand

Als Skriptvariable benötigen wir zwei Zählvariable z für die Zeilen der Überführungsfunktion und k für die abzukellernden Zeichen. Außerdem verwenden wir `zeile` für die aktuelle Zeile, um den Zugriff auf die einzelnen Elemente zu vereinfachen (die Überführungsfunktion ist ja eine verkettete Liste). Wir gehen mit einer Zählschleife die einzelnen Zeilen der Überführungsfunktion durch und setzen dabei jeweils die Variable `zeile`. Wenn Zustand, Keller- und Eingabezeichen übereinstimmen, dann kellern wir die entsprechenden Zeichen ab und geben den neuen Zustand zurück. Für das Abkellern verwenden wir eine `FOR`-Schleife. Es kann aber auch der Fall auftreten, dass kein Zeichen abzukellern ist. Für diesen Fall wäre der Endwert der `FOR`-Schleife Null und die Schleife würde von Eins bis Null rückwärts zählen. Um diese Fehlerquelle auszuschließen, packen wir die Zählschleife in eine einseitige Verzweigung und führen sie nur dann aus, wenn auch Zeichen zum Abkellern vorhanden sind. Wenn wir keinen Eintrag in der Überführungsfunktion finden, der zur der aktuellen Kombination passt, dann geben wir `false` zurück.

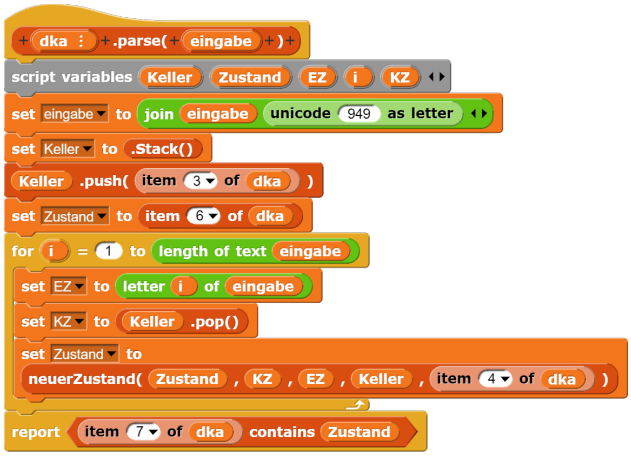

Abbildung 5.16: Methode parse()

Damit kann unser `parse`-Block ebenfalls sehr übersichtlich gestaltet werden. In der Tabelle unserer Überführungsfunktion finden wir zwei Zeilen, in denen das Kellerzeichen das Stapelvorbelegungszeichen # ist und das Eingabezeichen ϵ. Gemeint ist in beiden Fällen: Wenn die Eingabefolge abgearbeitet wurde und der Keller leer ist bis auf das Stapelvorbelegungszeichen, dann gehe in den akzeptierenden Zustand. Diese beiden Zeilen sind übrigens auch der Grund, warum AutoEdit den Kellerautomaten nicht für deterministisch hält. Bevor wir einen nichtdeterministischen Kellerautomaten implementieren, wenden wir einen kleinen Trick an: Wir ergänzen unsere Eingabefolge hinten um ein Epsilon und können damit die beiden Zeilen, mit denen wir in einen akzeptierenden Zustand gelangen, auch ausführen. Das Epsilon hat den Unicode 3B5. Umgerech-

net in eine Dezimalzahl liefert das 949 und so erhalten wir mit `unicode 949 as letter` ein Epsilon, das wir hinten an die Eingabe anfügen. Nun muss noch der Keller erzeugt werden, das Stapelvorbelegungszeichen abgekellert werden und der Zustand auf den Startzustand gesetzt werden. Damit sind die vorbereitenden Maßnahmen abgeschlossen und der eigentliche Parse-Vorgang kann beginnen. Wir haben das Eingabeband mit einer `FOR`-Schleife durch und lesen jeweils das Eingabezeichen ein, holen das Kellerzeichen aus dem Keller und rufen den Block `neuerZustand` auf, um den neuen Zustand zu berechnen (und den Keller zu aktualisieren).

Abschließend prüfen wir, ob der erreichte Zustand in der Liste der akzeptierenden Zustände aufgezählt ist, und geben das Ergebnis als Boolean aus.

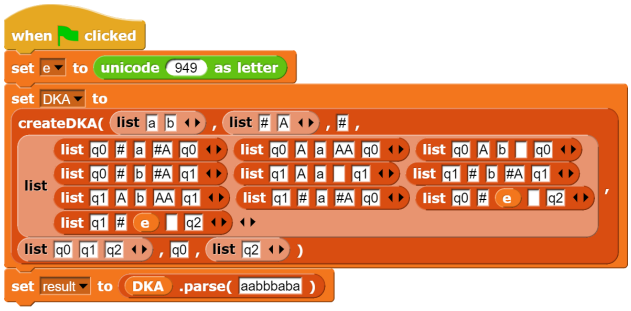

Abbildung 5.17: Skript Kellerautomat

Im Skript für den Automaten verwenden wir e als Hilfsvariable, um Epsilon leichter in die Überführungsfunktion einfügen zu können. Die Überführungsfunktion selbst ist als Tabelle aufgebaut, wobei in jeder Zeile zunächst der aktuelle Zustand steht, dann das Kellerzeichen, das Eingabezeichen, die abzukellernden Zeichen in umgekehrter Reihenfolge und schließlich der Folgezustand. Die Reihenfolge der abzukellernden Zeichen ergibt sich aus der Verwendung der Zählschleife im Block `neuerZustand`. Wir gehen die Zeichen von vorn beginnend durch: Zunächst wird das erste Zeichen abgekellert, dann das zweite usw. Im Zustandsgraphen des Automaten wird eine andere Reihenfolge benutzt: A# bedeutet, dass zunächst der Hashtag abgekellert wird und dann das A. Wenn wir diese Schreibweise verwenden möchten, müssen wir nur im Block `neuerZustand` in der `FOR`-Schleife für das Abkellern Start- und Endwert vertauschen.

Übrigens: unser Kellerautomat akzeptiert alle Worte aus a und b, die gleichviel a's wie b's enthalten.

5.7 Aufgaben

Aufgabe 5.1 *Gegeben sei die Grammatik* $G_2 = \{\{S, A, B\}, \{0, 1, 2\}, S \rightarrow AB, A \rightarrow 0A|0, B \rightarrow 1B2|12, S\}$. *Leiten Sie einige Worte dieser Sprache ab, indem Sie den entsprechenden Ableitungsbaum zeichnen. Beschreiben Sie, welche Sprache* G_2 *ist.*

Zeichnen Sie das Syntaxdiagramm von G_2.

Entwickeln Sie den zugehörigen Kellerautomaten.

Aufgabe 5.2 *Gegeben ist die Grammatik* $G_3 = \{\{S, A, B, C, Q, X, Y\}, \{a, b, c\}, S \rightarrow XY, X \rightarrow AQ|AB, Q \rightarrow XB, Y \rightarrow YC|c, A \rightarrow a, B \rightarrow b, C \rightarrow c, S\}$. *Bestimmen Sie die zugehörige Sprache und den Typ der Grammatik.*

Aufgabe 5.3 *Entwickeln Sie einen Kellerautomaten für die Sprache* $L_1 = \{0^n 1^{2n} | n > 0\}$. *Beispielwörter:*

011 wird akzeptiert

0011 wird nicht akzeptiert

000111111 wird akzeptiert

Hinweis: Versuchen Sie das Verhältnis von doppelt so vielen Einsen wie Nullen im Keller abzubilden.

Geben Sie eine Grammatik zu L_1 *an.*

Aufgabe 5.4 *Entwickeln Sie einen Kellerautomaten für die Sprache:* $L_2 = \{a^k b^n c^m | k, n, m > 0 \wedge k > n + m\}$. *Beispielwörter:*

aaabc wird akzeptiert, denn n=m=1, k=3, $1 + 1 < 3$

aabbcc wird nicht akzeptiert, n=m=k=2

aaaabbc wird akzeptiert, denn n=2, m=1, k=4, $2 + 1 < 4$

Hinweis: Der Automat soll sich die vorhandenen a's zunächst „merken", um dann die nachfolgenden Zeichen davon zu subtrahieren.

Geben Sie eine Grammatik zu L_2 *an.*

Aufgabe 5.5 *Gegeben ist die Sprache* $L_{10} = \{0^{m+2n}1^m | m > 0, n \geq 0\}$.

Geben Sie eine Grammatik zu L_{10} *an.*

Zeichnen Sie das Zustandsdiagramm des zugehörigen Kellerautomaten.