Automaten und Sprachen

Fritz Hasselhorn

15. September 2015

Inhaltsverzeichnis

1 Wozu endliche Automaten? 3

2 Alphabet, Wort und Sprache 5

3 Deterministische endliche Automaten (DEA) 7
3.1 Definition und Funktionsweise 7
3.2 Konstruktion von DEAs 9
3.3 Mindestens, höchstens und genau 11
3.4 Die Minimierung von endlichen Automaten 11

4 Nichtdeterministische endliche Automaten (NEA) 17
4.1 Funktionsweise und Definition des NEA 17
4.2 Die Potenzmengenkonstruktion 20
4.3 Die vereinfachte Potenzmengenkonstruktion 23

5 Programmierung von endlichen Automaten 26
5.1 Erste Version des DEA 26
5.2 Erste Verbesserung: Filterung der Eingabe 28
5.3 Überführungsfunktion als Tabelle 29

Kapitel 1

Wozu endliche Automaten?

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 endliche viele Eingaben und endliche viele 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, die Fahrtrichtung (aufwärts oder abwärts) und die Anfragen, die bisher noch nicht angefahren wurden.

Die Theorie der endlichen Automaten beschreibt, wie man solche Systeme entwirft. In der Informatik gibt es viele Beispiele für solche Systeme mit 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. Textbasierte 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.

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“.

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.*

Kapitel 2

Alphabet, Wort und Sprache

Rechner arbeiten im Prinzip mit Texten, d.h. mit Folgen von Symbolen aus einem bestimmten Alphabet. Auch graphische Programmiersprachen werden letztlich als Texte 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** λ ist die leere Buchstabenfolge. (Manchmal benutzt man die Bezeichnung ε statt λ).

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 **Sprache** L über einem Alphabet Σ ist eine Teilmenge von Σ^*.

Kapitel 3

Deterministische endliche Automaten (DEA)

3.1 Definition und Funktionsweise

Ein endlicher Automat 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 (auch Endzustände genannt). Außerdem gehört zum Automaten eine Überführungsfunktion, die zu jedem Zustand und zu jedem Eingabezeichen den Folgezustand angibt.

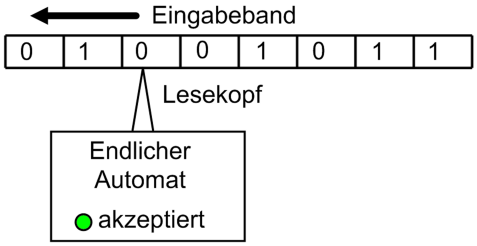

Abbildung 3.1: Aufbau eines DEA

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

der endliche Automat, der sich im Zustand s_i befindet und das Eingabezeichen e_k liest, in den Zustand $s_j = u(s_i, e_k)$ über und bewegt seinen Lesekopf um ein Eingabezeichen nach rechts. Ist das Wort auf dem Eingaband abgearbeitet und der Automat befindet sich in einem akzeptierender 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.

Ein endlicher Automat kann mit einem Zustandsdiagramm beschrieben werden. Ein Zustandsdiagramm ist ein gerichteter Graph. Die Knoten (Kreise) entsprechen den Zuständen. Der Startzustand ist mit einem zusätzlichen Pfeil auf den Kreis gekennzeichnet (hier q0). Die akzeptierenden Zustände sind durch einen Doppelkreis gekennzeichnet (hier q0 und q2). Die Eingaben stehen an den Kanten (Pfeilen). Der Pfeil von q0 nach q1 ist mit „1“ beschriftet. Das bedeutet: „Befindet sich der Automat im Zustand q0, dann geht er durch Eingabe von 1 in den Zustand q1“.

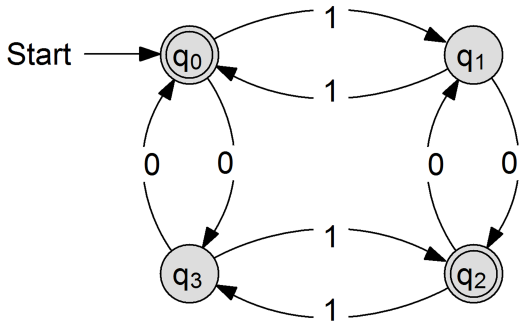

Abbildung 3.2: Beispiel eines DEA

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

An der tabellarischen Darstellung der Übergangsfunktion, die AutoEdit auf der Exportieren-Seite anzeigt, erkennt man, ob die Übergangsfunktion eindeutig und vollständig ist. Die Eindeutigkeit sieht man daran, dass in jeder Zelle der Tabelle nur ein Folgezustand steht. Ohne Eindeutigkeit ist der Automat nicht deterministisch. Die Vollständigkeit zeigt sich daran, dass

alle Zelle der Tabelle ausgefüllt sind. In der Regel verlangen wir von deterministischen endlichen Automaten auch die Vollständigkeit. Teilweise wird in Aufgabenstellungen aber ausdrücklich auf die Darstellung der Fehlerzuständer und der Übergänge in die Fehlerzustände verzichtet.

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

1. einer endlichen Menge von Zuständen Q,
2. einem Eingabealphabet Σ,
3. einer Überführungsfunktion δ, die zu Zustand und Eingabezeichen den Folgezustand berechnet,
4. einem Startzustand (hier q0),
5. einer endlichen Menge von akzeptierenden Zuständen E (Der häufig verwendete Begriff „Endzustände“ ist mißverständlich, weil der Automat 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 nicht. 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.

3.2 Konstruktion von DEAs

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

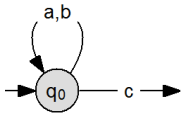

Abbildung 3.3: 0 bis n Wiederholungen

In der Abbildung können a und b beliebig oft wiederholt werden. Es ist aber nicht sichergestellt, dass a und b überhaupt wiederholt werden, weil man auch gleich den Übergang mit c nehmen kann. Wenn man sicherstellen will, dass mindestens eine Wiedeholung erfolgt, muss man einen Übergang davorsetzen:

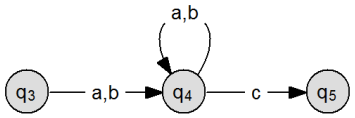

Abbildung 3.4: 1 bis n Wiederholungen

Ein weiterer Grundbaustein ist die Verzweigung:

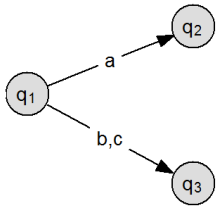

Abbildung 3.5: Verzweigung

Hier kann entweder mit a, b bei q3 fortgesetzt werden oder mit c bei q2.

Die Verzweigung wird manchmal verbunden mit einem absorbierenden Fehlerstand, auch Falle (englisch *trap*) genannt. Der Fehlerzustand q1 heißt absorbierend, weil man ihn nicht mehr verlassen kann. Im Beispiel sind Worte gesucht, die an einer bestimmten Stelle ein a haben. Tritt dort ein b oder c auf, dann gehört das Wort nicht zur Sprache des DEA.

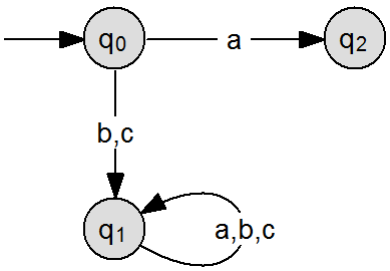

Abbildung 3.6: absorbierender Fehlerzustand

Ein endlicher Automat kann nur zählen, indem er in den nächsten Zustand geht.

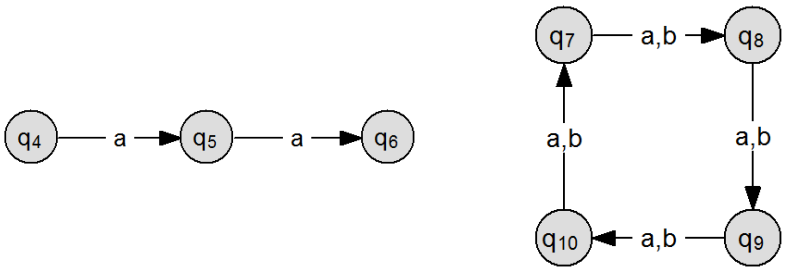

Abbildung 3.7: Fortlaufendes und wiederholtes Zählen

3.3 Mindestens, höchstens und genau

3.4 Die Minimierung von endlichen Automaten

Bei Aufgaben zur endlichen Automatten gibt es häufig verschiedene Lösungen, die sich bisweilen auch in der Zahl der benötigten Zustände unterscheiden. Dies wirft die Frage aus, ob es zu einer gegegenen Sprache einen endlichen Automaten mit weniger Zuständen gibt. Weniger Zustände bedeuten weniger Aufwand. Wie finden wir zu einner Sprache einen minimalen endlichen Automaten?

Es ist klar, dass alle Zustände, die vom Startzustand aus nicht erreichbar sind, nicht in einen minismalen endlichen Automaten gehören (Beispiele dafür finden sich im Kapitel 4.2 Potenzmengenkonstruktion).

Zur Minimierung eines endichen Automaten müssen Zustände zusammengelegt werden. Dabei können Zustände, die keine akzptierenden Zustände sind, und akzeptierende Zustände nicht zusammengelegt werden.

Wenn man mit dem gleichen Eingabezeichen von zwei Zuständen q1 und q2 in zwei Zustände q1' und q2' gelangt und q1' und q2' nicht zusammengelegt werden können, dann können auch q1 und q2 nicht zusammengelegt werden.

Ein endlicher Automat A heißt **minimal**, wenn es keinen endlichen Automaten mit weniger Zuständen als A gibt, der die gleiche Sprache akzeptiert.

Aus diesen Ideen ergibt sich der folgende Algorithmus zur Bestimmung eines minimalen endlichen Automaten:

1. Entferne alle Zustände, die vom Startzustand aus nicht erreicht werden können.

2. Stelle eine Tabelle für alle möglichen Paare von Zuständen auf und markiere darin alle Paare $(p_i|p_i)$ als äquivalent.

3. Markiere alle Paare, bei denen genau ein Zustand zu den akzeptierenden Zuständen gehört und der andere nicht, als nicht zusammenlegbar.

4. Wiederhole

   > markiere alle Paare, für die es ein Eingabezeichen a gibt, so dass die mit a erreichten Folgezustände bereits markiert wurden, als nicht zusammenlegbar,

   bis in einem Durchgang keine Änderungen vorgenommen wurden.

5. Lege alle nicht markierten Paare zusammen.

Wir wählen das folgende Beispiel für die Minimierung eines Automaten:

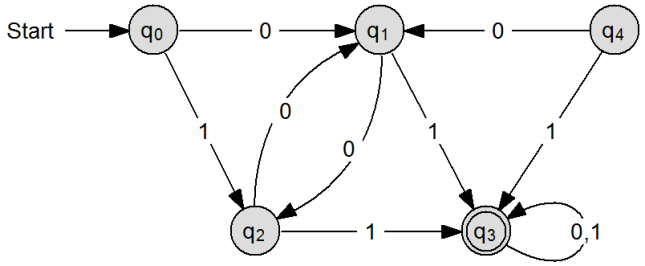

Abbildung 3.8: Beispiel-DEA zur Minimierung

Daraus ergibt sich die Übergangsfunktion:

δ	0	1
q_0	q_1	q_2
q_1	q_2	q_3
q_2	q_1	q_3
q_3	q_3	q_3
q_4	q_1	q_3

Im Zustand q4 kommt kein Übergang an, er kann deshalb weggelassen werden. Für die restlichen Zustände stellen wir eine Tabelle auf, in der wir die gleichnamigen Paare als äquivalent markieren. Da die Reihenfolge der Zustände keine Rolle spielt, reicht uns eine Hälfte der Tabelle:

q_0	$\equiv$			
q_1		$\equiv$		
q_2			$\equiv$	
q_3e				$\equiv$
	q_0	q_1	q_2	q_3e

Wir markieren alle Paare aus akzeptierenden und nichtakzeptierenden Zuständen als nicht zulässig:

q_0	$\equiv$			
q_1		$\equiv$		
q_2			$\equiv$	
q_3e	x	x	x	$\equiv$
	q_0	q_1	q_2	q_3e

Wir prüfen, ob die restlichen Paare in einen bereits markierten Zustand übergehen:

$(q_0|q_1) \xrightarrow{0} (q_1|q_2)$ Ziel ist noch frei.

$(q_0|q_1) \xrightarrow{1} (q_3|q_2)$ Ziel ist markiert, also muss das Ausgangspaar markiert werden.

$(q_0|q_2) \xrightarrow{0} (q_1|q_1)$ Ziel ist äquivalent.

$(q_0|q_2) \xrightarrow{1} (q_3|q_2)$ Ziel ist markiert, also muss das Ausgangspaar markiert werden.

q_0	$\equiv$			
q_1	x	$\equiv$		
q_2	x		$\equiv$	
q_3 e	x	x	x	$\equiv$
	q_0	q_1	q_2	q_3e

Da im ersten Durchgang zwei Paare neu markiert wurden, wird der Schritt wiederholt.

2. Durchgang:

Wir prüfen $(q_1|q_2)$:

$(q_1|q_2) \xrightarrow{0} (q_2|q_1)$ Ziel ist noch frei.

$(q_1|q_2) \xrightarrow{1} (q_3|q_3)$ Ziel ist äquivalent, also wird das Ausgangspaar nicht markiert.

q_0	$\equiv$			
q_1	x	$\equiv$		
q_2	x		$\equiv$	
q_3e	x	x	x	$\equiv$
	q_0	q_1	q_2	q_3e

Da im 2. Durchgang keine neuen Paare markiert wurden, wird das Verfahren abgebrochen.

Wir kontrollieren in der Tabelle der Übergangsfunktion:

δ	0	1
q_1	q_2	q_3
q_2	q_1	q_3

Wenn q_1 gleich q_2 ist, werden die beiden Zeilen gleich. Deshalb können q_1 und q_2 zusammengelegt werden.

Wir erhalten die folgende Tabelle für den minimierten Automaten:

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

Der Automat akzeptiert alle Worte mit mindestens zwei Buchstaben, die abgesehen vom Anfangsbuchstaben mindestens eine 1 enthalten.

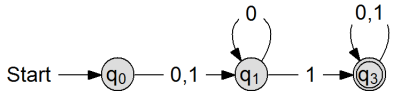

Abbildung 3.9: Minimierter DEA

Als zweites Beispiel für die Minimierung wählen wir folgenden Automaten:

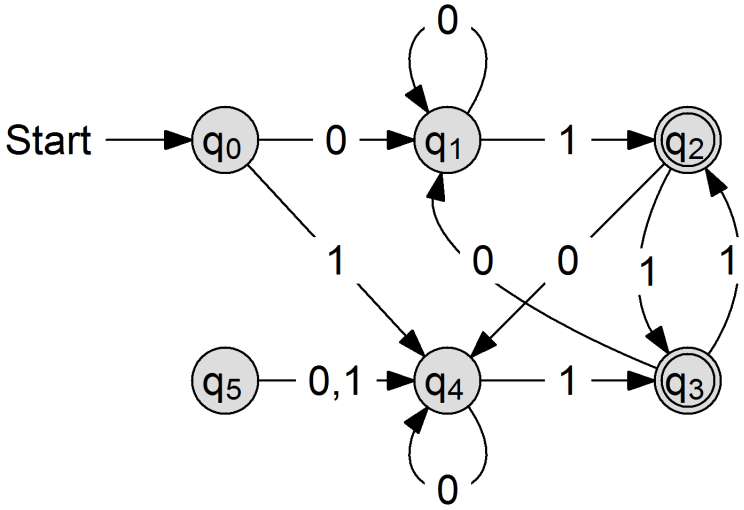

Abbildung 3.10: 2. Beispiel zur Minimierung

δ	0	1
q_0	q_1	q_4
q_1	q_1	q_2
q_2	q_4	q_3
q_3	q_1	q_2
q_4	q_4	q_3
q_5	q_4	q_4

Der Zustand q_5 ist vom Startzustand aus nicht erreichbar, wir können ihn weglassen. Damit ergibt sich folgende Tabelle:

q_0	$\equiv$				
q_1		$\equiv$			
q_2 e			$\equiv$		
q_3 e				$\equiv$	
q_4					$\equiv$
	q_0	q_1	q_2 e	q_3 e	q_4

Wir markieren alle Paare aus akzeptierenden und nichtakzeptierenden Zuständen als nicht zulässig:

q_0	$\equiv$				
q_1		$\equiv$			
q_2 e	x	x	$\equiv$		
q_3 e	x	x		$\equiv$	
q_4			x	x	$\equiv$
	q_0	q_1	q_2 e	q_3 e	q_4

Wir prüfen, ob die restlichen Paare in einen bereits markierten Zustand übergehen:

$(q_0|q_1) \xrightarrow{0} (q_1|q_1)$ Ziel ist äquivalent.

$(q_0|q_1) \xrightarrow{1} (q_4|q_2)$ Ziel ist markiert, also muss das Ausgangspaar markiert werden.

$(q_2|q_3) \xrightarrow{0} (q_4|q_1)$ Ziel ist noch frei.

$(q_2|q_3) \xrightarrow{1} (q_3|q_2)$ Ziel ist noch frei, also muss Ausgangspaar vorläufig nicht markiert werden.

$(q_0|q_4) \xrightarrow{0} (q_1|q_4)$ Ziel ist noch frei.

$(q_0|q_4) \xrightarrow{1} (q_4|q_3)$ Ziel ist markiert, also muss das Ausgangspaar markiert werden.

$(q_1|q_4) \xrightarrow{0} (q_2|q_3)$ Ziel ist noch frei.

$(q_1|q_4) \xrightarrow{1} (q_4|q_3)$ Ziel noch frei, also muss Ausgangspaar vorläufig nicht markiert werden.

Das ergibt sich noch dem ersten Durchgang folgende Tabelle:

q_0	$\equiv$				
q_1	x	$\equiv$			
q_2 e	x	x	$\equiv$		
q_3 e	x	x		$\equiv$	
q_4	x		x	x	$\equiv$
	q_0	q_1	q_2 e	q_3 e	q_4

Wir untersuchen die freien Felder:

$(q_2|q_3) \xrightarrow{0} (q_4|q_1)$ Ziel ist noch frei.

$(q_2|q_3) \xrightarrow{1} (q_3|q_2)$ Ziel ist noch frei, also muss Ausgangspaar vorläufig nicht markiert werden.

$(q_1|q_4) \xrightarrow{0} (q_2|q_3)$ Ziel ist noch frei.

$(q_1|q_4) \xrightarrow{1} (q_4|q_3)$ Ziel noch frei, also muss Ausgangspaar vorläufig nicht markiert werden.

Im zweiten Durchgang wurden keine neuen Paare notiert. Damit ist der Algorithmus abgeschlossen. Die Zustände q_1 und q_4 sind äquivalent und können zusammengefasst werden. Die Zustände q_2 und q_3 sind ebenfalls äquivalent und können zusammengefasst werden.

Wir überprüfen das Ergebnis an der Übergangstabelle:

δ	0	1
q_0	q_1	q_4
q_1	q_1	q_2
q_2	q_4	q_3
q_3	q_1	q_2
q_4	q_4	q_3

In beiden Fällen sind die Zeilen gleich, wenn gilt $q_1 \equiv q_4$ und $q_2 \equiv q_3$. Damit erhalten wir als minimierten endlichen Automaten:

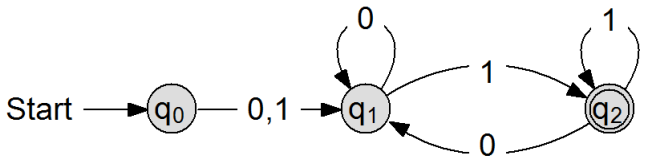

Abbildung 3.11: 2. Beispiel zur Minimierung

Der DEA akzeptiert alle Worte, die auf 1 enden und mindestens aus zwei Buchstaben bestehen.

Kapitel 4

Nichtdeterministische endliche Automaten (NEA)

4.1 Funktionsweise und Definition des NEA

Die Forderung nach der Determiniertheit, also nach der Eindeutigkeit endlicher Automaten erschneit 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 nicht gelöst werden können, für die aber nichtdeterministische Lösungsverfahren existieren. Es ist leichter, den Begriff „Nichtdeterminismus“ im einfachen Kontext von endlichen Automaten einzuführen als in komplexen Zusammenhängen.

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.

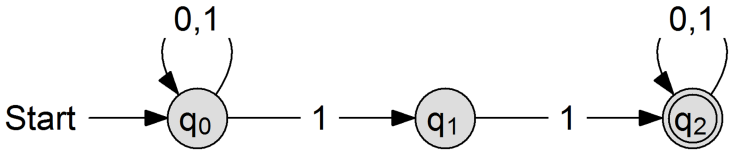

Abbildung 4.1: 1. Beispiel eines NEA

Das erste Beispiel zeigt einen nichtdeterministischen endlichen Automaten (NEA), der 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 „1“ beschriftet sind.

δ	0	1
q_0	q_0	q_0, q_1
q_1	–	q_2
q_2	q_2	q_2

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 dsa 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.

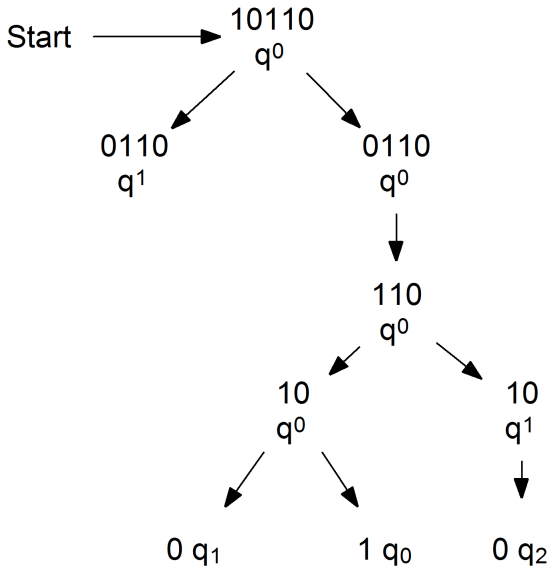

Abbildung 4.2: Entscheidungsbaum für 10110

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

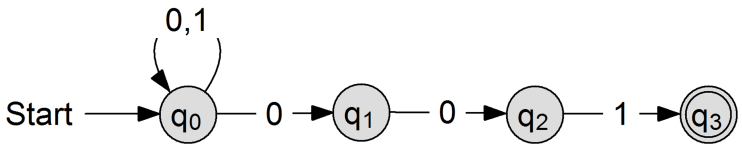

Abbildung 4.3: 2. Beispiel eines NEA

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

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

Definition:
Ein **nichtdeterministischer endlicher Automat (NEA)** besteht aus

1. einer endlichen Menge von Zuständen Q,
2. einem Eingabealphabet Σ,
3. einer Überführungsfunktion δ, die zu Zustand und Eingabezeichen den Folgezustand berechnet, wobei δ weder eindeutig noch vollständig sein muss,
4. einem Startzustand (hier q0),
5. einer endlichen Menge von akzeptierenden Zuständen E.

4.2 Die Potenzmengenkonstruktion

Wie kann man nun einen nichtdeterministischen endlichen Automaten durch einen deterministischen endlichen Automaten simulieren? Von der Definition her besteht der einzige Unterschied zwischen den beiden Typen in der Überführungsfunktion, weil in der Tabelle der Übergangsfuntion 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. Wenn gilt: $M = \{A, B, C\}$, dann ist

$P(M) = \{\{A, B, C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A\}, \{B\}, \{C\}, \emptyset\}$

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 Binomialkoeffizienten. 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. Bei einer vierelementigen Menge gibt es eine Teilmenge mit vier Elementen, vier Teilmengen mit drei Elementen, sechs Teilmengen mit zwei Elementen, vier Teilmengen mit einem Element und eine leere Teilmenge, also 16 verschiedene Teilmengen.

Wir betrachten nochmals die Überführungsfunktion des ersten Beispiel-NEAs:

NEA δ	0	1
q_0	q_0	q_0, q_1
q_1	$-$	q_2
q_2	q_2	q_2

Dabei war q2 der akzeptierende Zustand.

Um etwas Schreibarbeit zu sparen, schreiben wir die Teilmengen mit jeweils einem q und einem Index, also nicht $\{q_0, q_1, q_2\}$, sondern $\{q_{012}\}$. Damit besteht die Potenzmenge aus den Elementen $\{q_{012}\}$, $\{q_{01}\}$, $\{q_{02}\}$, $\{q_{12}\}$, $\{q_0\}$, $\{q_1\}$, $\{q_2\}$, $\emptyset$. Dabei gehören alle Teilmengen zu den akzeptierenden Zuständen, die q2 enthalten.

Die Zeilen für das neue δ erhalten wir, indem wir jeweils die entsprechenden Zeilen 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. Akzeptierende Zustände sind q_{012}, q_{02}, q_{12} und q_2.

DEA δ	0	1
q_{012}	q_{02}	q_{012}
q_{01}	q_0	q_{012}
q_{02}	q_{02}	q_{012}
q_{12}	q_2	q_2
q_0	q_0	q_{01}
q_1	$\emptyset$	q_2
q_2	q_2	q_2
$\emptyset$	$\emptyset$	$\emptyset$

Umgesetzt in einen Zustandsgraphen ergibt sich das folgende Bild:

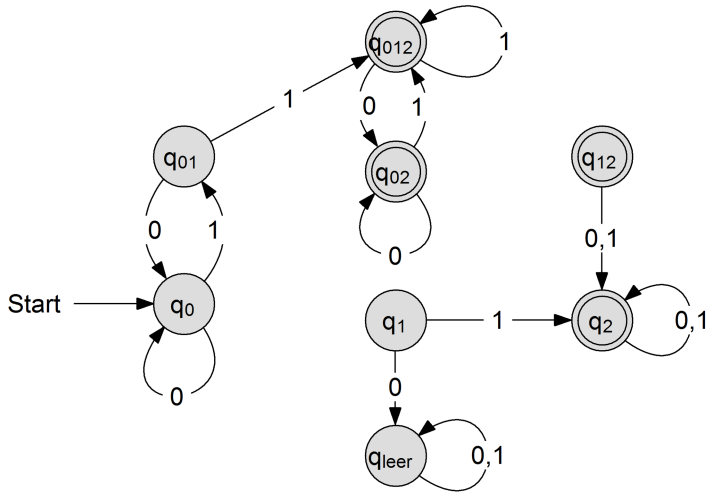

Abbildung 4.4: Potenzmenge des 1. Beispiels

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

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

NEA δ	0	1
q_0	q_0, q_1	q_0
q_1	q_2	–
q_2	–	q_3
q_3 e	–	–

Hieraus ergibt sich folgendes δ für die Potenzmenge:

DEA δ	0	1
q_{0123} e	q_{012}	q_{03}
q_{012}	q_{012}	q_0
q_{013} e	q_{012}	q_0
q_{023} e	q_{01}	q_{03}
q_{012}	q_{02}	q_{012}
q_{01}	q_{012}	q_0
q_{23} e	$\emptyset$	q_3
q_{012}	q_{02}	q_{012}
q_{02}	q_{01}	q_{03}
q_{13} e	q_2	$\emptyset$
q_{03} e	q_{01}	q_0
q_{12}	q_2	q_3
q_0	q_{01}	q_0
q_1	q_2	$\emptyset$
q_2	$\emptyset$	q_3
q_3 e	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

Daraus ergibt sich der folgende Zustandsgraph:

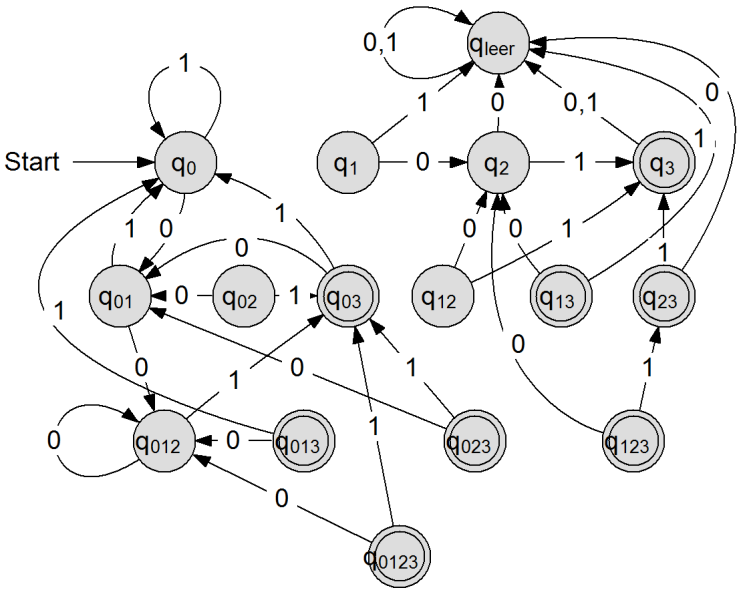

Abbildung 4.5: Potenzmenge des 2. Beispiels

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 q_{02}, q_{013}, q_{023} und q_{0123}. Damit reduziert sich der relevante Teil der Zustandsgraphen auf die vier Zustände q_0, q_{01}, q_{03} und q_{012}.

Die 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.

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, wobei der akzeptierende Zustand mit „e“ gekennzeichnet wird. Neuauftretende Kombinationen werden unterstrichen:

NEA δ	0	1
q_0	q_0	q_0, q_1
q_1	–	q_2
q_2 e	q_2	q_2

Für den DEA übernehmen wir zunächst die erste Zeile:

DEA δ	0	1
q_0	q_0	$\underline{\underline{q_{01}}}$

Rechts taucht die neue Kombination q_{01} auf, die wir auf die linke Seite übernehmen. Für die rechte Seite fassen wir die Zeilen von q_0 und q_1 aus der NEA-Tabelle zusammen:

DEA δ	0	1
q_0	q_0	q_{01}
q_{01}	q_0	$\underline{\underline{q_{012}}}$

Rechts taucht die neue Kombination q_{012} auf, die wir wieder auf die linke Seite übernehmen. Da die Kombination den akzeptierenden Zustand q_2 enthält, kennzeichen wir sie mit „e“. Für die rechte Seite fassen wir die Zeilen von q_0, q_1 und q_2 aus der NEA-Tabelle zusammen:

DEA δ	0	1
q_0	q_0	q_{01}
q_{01}	q_0	q_{012}
q_{012} e	$\underline{\underline{q_{02}}}$	q_{012}

Die neue Kombination q_{02} wird übernommen:

DEA δ	0	1
q_0	q_0	q_{01}
q_{01}	q_0	q_{012}
q_{012} e	q_{02}	q_{012}
q_{02} e	q_{02}	q_{012}

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:

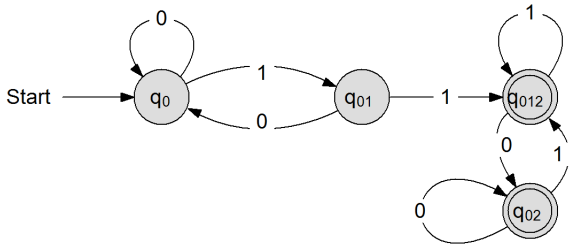

Abbildung 4.6: DEA zum 1. Beispiel eines NEA

Entsprechend bearbeiten wir die Tabelle für das zweite Beispiel:

NEA δ	0	1
q_0	q_0, q_1	q_0
q_1	q_2	–
q_2	–	q_3
q_3 e	–	–

Wir erhalten für den zugehörigen DEA:

NEA δ	0	1
q_0	$\underline{\underline{q_{01}}}$	q_0
q_{01}	$\underline{\underline{q_{012}}}$	q_0
q_{012}	q_{012}	$\underline{\underline{q_{03}}}$
q_{03} e	q_{01}	q_0

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

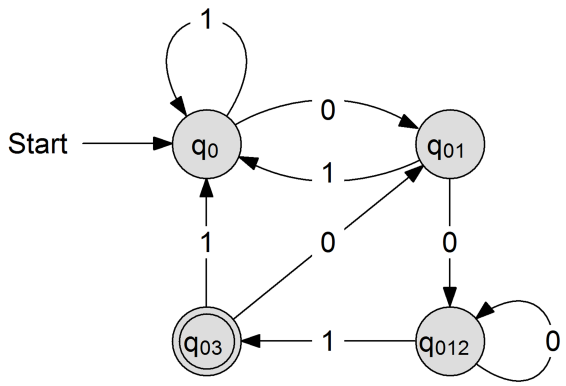

Abbildung 4.7: DEA zum 2. Beispiel eines NEA

Kapitel 5

Programmierung von endlichen Automaten

Der folgende deterministische endliche Automat soll in BYOB implementiert werden:

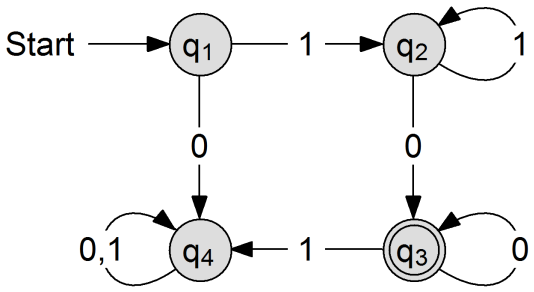

Abbildung 5.1: DEA 1+0+

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.

Man sieht, dass Delta überwiegend aus geschachtelten Verzweigungen besteht. Zunächst müssen die vier Zustände unterschieden werden und dann das jeweilige Eingabezeichen. Da wir hier nur zwei Eingabezeichen haben, reicht zur Unterscheidung der Eingabezeichen jeweils eine IF-ELSE-Schleife aus. Bei drei und mehr Eingabezeichen müsste für jedes Eingabezeichen eine eigene IF-Schleife eingesetzt werden, z.B.

Abbildung 5.2: Überführungsfunktion

```
if Zustand = 0
 if EZ = 0
   set Folgezustand to ...
 if EZ = 1
   set Folgezustand to ...
 if EZ = 2
   set Folgezustand to ...
```

Nur beim Zustand q_4 vereinfacht sich die Funktion, weil dort alle Eingaben wieder nach q_4 führen.

Eine Alternative bestünde darin, dass überall dort, wo der Folgezustand gesetzt wird (`set Folgezustand to`), der Folgezustand mit `report` gleich zurückgegeben wird.

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 5.3: Eingabe

Für unseren DEA benötigen wir nun noch drei globale Variable: den Zu-

stand, das Wort und eine Zählvariable i, weil wir das Wort zeichenweise durchgehen müssen. Das Hauptprogramm setzt den Zustand auf 1 (Startzustand), i ebenfalls auf 1 und das Wort auf die eingegebene Zeichenfolge. Dann wird für jeden Buchstaben `(letter(i)of(wort))`aus Eingabezeichen und Zustand der Folgezustand bestimmt. Zum Schluss überprüft das Programm, ob der akzeptierende Zustand q_3 erreicht wurde und macht eine entsprechende Ausgabe:

Abbildung 5.4: Skript des DEA

5.2 Erste Verbesserung: Filterung der Eingabe

In der jetzigen Fassung ist der Automat nicht gegen Fehleingaben gesichert. Z.B. wird das Wort „120“ akzeptiert, obwohl 2 nicht zu den gültigen Eingabezeichen gehört. Nun könnte man in die Eingabe eine Abfrage einbauen, ob das Wort nur aus Einsen und Nullen besteht. Wir wählen hier einen anderen Weg, der sich stärker an der Definition des DEA orientiert und erstellen eine neue Variable Eingabealphabet, in der wir die Eingabezeichen als Liste speichern:

Innerhalb unseres Reporters „Eingabe“ geben 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.

Wenn ein anderes Eingabealphabet vorliegt, muss lediglich die Liste „Eingabealphabet“ entsprechend ergänzt werden.

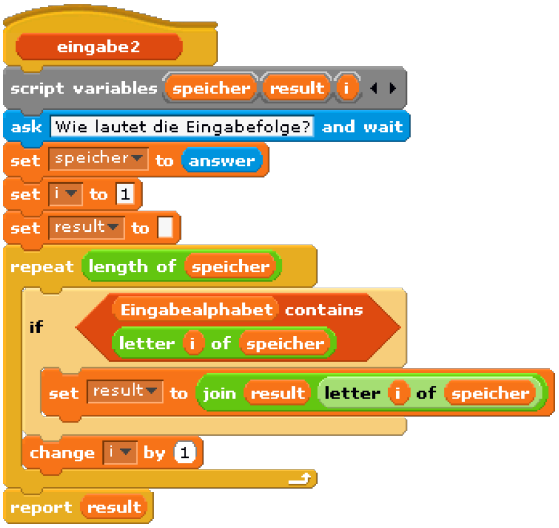

Abbildung 5.5: Eingabe mit Prüfung

5.3 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 als Tabelle:

δ	0	1
q_1	q_4	q_2
q_2	q_3	q_2
q_3	q_3	q_4
q_4	q_4	q_4

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

An dieser Stelle wird deutlich, warum wir die Zustände mit Eins beginnend durchnummeriert haben. Wenn wir auf das Q verzichten, also die Zustände nur mit 1, 2, 3, 4 bezeichnen, dann gibt der Zustand direkt die Zeile an. Bei den Eingabezeichen brauchen wir bei diesem Eingabealphabet nur 1 addieren, um die richtige Spalte zu verhalten.

Um die Tabelle der Überführungsfunktion übersichtlich zu halten, speichern wir sie als Liste, wobei wir die Zustände mit 1, 2, 3, 4 bezeichnen

und jede Zeile als ein Wort speichern. Dazu fügen wir im Hauptprogramm folgende Zeile ein:

Mit Hilfe der globalen Variablen Tabelle lässt sich Delta dann wesentlich einfacher implementieren:

Abbildung 5.6: Überführungsfunktion mit Tabelle

Diese Version des DEA lässt sich problemlos an einen anderen endlichen Automaten anpassen. Wir müssen lediglich das Eingabealphabet verändern, die Tabelle für die neue Überführungsfunktion einsetzen und ggf. die Abfrage der akzeptierenden Zustände ändern. Wenn das Eingabealphabet sich nicht mehr so leicht in Spalten umrechnen lässt (z.B. wenn Buchstaben vorkommen), brauchen wir noch einen Reporter, der zu jedem Eingabezeichen die richtige Spalte liefert.