Einführung in Formale Sprachen, Berechenbarkeit, Informations- und Lerntheorie

von
Norbert Blum

Oldenbourg Verlag München Wien

Prof. Dr. Norbert Blum lehrt an der Rheinischen Friedrich-Wilhelms-Universität Bonn, Institut für Informatik.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
oldenbourg.de

Lektorat: Margit Roth
Herstellung: Anna Grosser
Coverentwurf: Kochan & Partner, München
Coverausführung: Gerbert-Satz, Grasbrunn
Gedruckt auf säure- und chlorfreiem Papier
Gesamtherstellung: Druckhaus „Thomas Müntzer“ GmbH, Bad Langensalza

ISBN 3-486-27433-3
ISBN 978-3-486-27433-2

Inhaltsverzeichnis

Vorwort

Was ist Theoretische Informatik? Insbesondere, was sollten die Inhalte einer Einführung in die Theoretische Informatik sein? Diese Fragen habe ich mir vor zehn Jahren, als ich mein erstes Buch über Theoretische Informatik schrieb, gestellt. Was die Inhalte einer grundlegenden einführenden Vorlesung im Bachelor-Studiengang anbetrifft, hat sich wenig geändert. Viele Geräte im täglichen Gebrauch enthalten so genannte Computer, die nicht annähernd die Fähigkeit eines PC's besitzen. Man denke zum Beispiel nur an eine Aufzugssteuerung. Hierfür genügt bereits ein endlicher Automat. Eine grundlegende Einführung in die Theorie „einfacher" Automaten und den mit diesen eng verknüpften formalen Sprachen gehört in den Bachelor-Studiengang Informatik und wird im ersten Kapitel behandelt. Kann man im Prinzip jede Funktion berechnen? Falls nein, kann man die Klasse der berechenbaren Funktionen formal charakterisieren? Auf diese Frage sollte jeder Informatiker eine Antwort wissen. Darüber hinaus haben die zur Beantwortung dieser Fragen entwickelten Techniken auch anderswo ihre Anwendung gefunden. Darum behandelt das zweite Kapitel die theoretische Berechenbarkeit von Funktionen. Eine Funktion ist auch dann theoretisch berechenbar, wenn jedes effektive Rechenverfahren mindestens eintausend Jahre benötigt, um den Funktionswert $f(x)$ zu berechnen. Eine derartige Funktion ist ganz gewiß nicht praktisch berechenbar. Diese Beobachtung führt zum Begriff der praktischen Berechenbarkeit, die im dritten Kapitel des Buches behandelt wird. Als praktisch berechenbar wird eine Funktion f angesehen, wenn es ein Computerprogramm gibt, das für ein beliebiges x aus dem Definitionsbereich den Funktionswert $f(x)$ auf einem Rechner in Polynomzeit berechnet. Für viele Funktionen, die in der Praxis auftreten, kennt man zwar kein derartiges Computerprogramm, jedoch kann man bei gegebenem x und gegebener Herleitung für $f(x)$ in polynomieller Zeit überprüfen, ob die Herleitung korrekt ist. Dies führt zur Begriffsbildung der NP-vollständigen Probleme, die in gewissem Sinn „schwierigste" derartige Probleme sind. In der Tat sind auch viele Funktionen der Praxis NP-vollständig und verlangen eine andere Behandlung als Probleme, für die bekannt ist, dass sie praktisch berechenbar sind. Neben der Theorie der NP-Vollständigkeit werden auch grundlegende Themen der Kryptographie, die im direkten Zusammenhang mit praktisch nicht berechenbaren Funktionen stehen, behandelt.

Zu den grössten Herausforderungen unserer Zeit gehört die Erforschung von Information in Bezug auf ihrer Generierung, ihre Extraktion und ihrer Behandlung. Daher ist

der zweite Teil des Buches einer Einführung in die Informations- und Lerntheorie gewidmet. Kapitel 4 führt in die so genannte klassische Informationstheorie ein. Neuere Entwicklungen bezüglich der Extraktion von Information setzen die algorithmische Informationstheorie voraus. Daher behandelt das fünfte Kapitel die algorithmische Informationstheorie. Um Gesetzmäßigkeiten aus beobachteten Daten ableiten zu können, muss man zunächst die Regelmäßigkeiten in den Daten erkennen. Das Gegenteil von Strukturiertheit in Daten ist die Zufälligkeit. Daher führt Kapitel 6 in die Theorie der Zufallsfolgen ein. Systeme, die aus beobachteten Daten Gesetzmäßigkeiten ableiten, heißen induktive Inferenzsysteme. Kapitel 7 behandelt diese. Schließlich gibt das achte Kapitel eine Einführung in die Theorie des Lernens von Konzepten.

Das Buch ist an Personen gerichtet, die über grundlegende Kenntnisse der Analysis und der linearen Algebra, wie sie in den Grundlagenvorlesungen über Analysis und Lineare Algebra vermittelt werden, verfügen. Insofern wurde auf einen Anhang „Mathematische Grundlagen“ verzichtet. Beweise werden, mit Ausnahme der Beweise von Behauptungen und Lemmata, die selbst Teil eines Beweises sind, mit ■ abgeschlossen. Deren Ende wird mit □ angezeigt. Beispiele, die noch fortgeführt werden, sind mit ◊ abgeschlossen. Ansonsten ist das Ende eines Beispiels mit ♦ markiert. Das Buch enthält Übungsaufgaben, die zum Teil dort im Text stehen, wo sie bearbeitet werden sollen. Weitere Übungsaufgaben stehen jeweils am Kapitelende. Schwere Übungsaufgabe sind mit * und sehr schwere Übungsaufgaben mit ** markiert. Um die Vertiefung in den bearbeiteten Teilbereichen zu vereinfachen, werden ausführliche Literaturhinweise gegeben. Bei der Vielzahl von ausgezeichneter Literatur habe ich sicherlich das eine oder andere gute Buch übersehen. Dies bitte ich zu entschuldigen.

Die Inhalte der ersten drei Kapitel sind für eine grundlegende Theorievorlesung im Bachelorstudiengang konzipiert. Die Kapitel 4 bis 8 könnten Gegenstand einer einführenden Veranstaltung über Informations- und Lerntheorie sein. Diese sind aus Vorlesungen, die ich an der Universität Bonn gehalten habe, entstanden.

Herr Christian Dorau und Herr Heinz Christian Steinhausen haben sich derart gründlich auf ihre Diplomprüfung über Informationstheorie vorbereitet, dass sie mir wertvolle Hinweise geben konnten. Dafür danke ich ihnen. Ich danke Herrn Mathias Hauptmann, der die Kapitel 4 bis 8 gelesen hat. Seine wertvollen Hinweise haben zur Verbesserung des Buches in vielerlei Hinsicht beigetragen. Des Weiteren danke ich Herrn Matthias Kretschmer für die Erstellung der Bilder. Dem Oldenbourg Verlag und insbesondere der Lektorin, Frau Margit Roth gilt mein Dank für die gute Zusammenarbeit.

Norbert Blum

Inhaltsverzeichnis

1 Automatentheorie und Formale Sprachen

Im täglichen Leben spielen Computer eine immer größer werdende Rolle. Viele Geräte für den täglichen Gebrauch enthalten so genannte Computer. Diese haben allerdings in der Regel nicht annähernd die Fähigkeiten eines PC's. So wird man für eine Aufzugssteuerung keinen PC verwenden, da man diese mit einem Automaten, der weitaus geringere Fähigkeiten besitzt, realisieren kann. Ein Aufzug muss lediglich wissen, wo er sich gerade befindet und welche Stockwerke anzufahren sind, da dort oder im Aufzug die betreffenden Knöpfe gedrückt wurden. Diesbezüglich sind nur endlich viele Konfigurationen möglich, so dass eine Aufzugssteuerung durch einen "endlichen Automaten" durchgeführt werden kann. Möchte man zum Beispiel in einem Taschenrechner arithmetische Ausdrücke auswerten, dann gibt es potentiell unendlich viele Konfigurationen, so dass die Auswertung arithmetischer Ausdrücke nicht durch einen endlichen Automaten durchführbar ist. Hierzu benötigt man einen unendlichen Speicher. Jedoch genügt ein einfacher Speicher, der nach dem Kellerprinzip arbeitet. Dies bedeutet, dass alles, was über dem zu lesenden Symbol im Keller steht, gelöscht werden muss, bevor dieses Symbol gelesen werden kann. Ein endlicher Automat, dem man einen Keller als unendlichen Speicher hinzufügt, heißt Kellerautomat. Ziel dieses Kapitels ist die Einführung von endlichen und von Kellerautomaten. Beide Konzepte werden zur Realisierung eines *Übersetzers (Compilers)* benötigt. Daher werden wir uns endliche und Kellerautomaten nahe bringen, indem wir die Prinzipien und Techniken, die für die Konstruktion von Übersetzern benötigt werden, erarbeiten. Dabei werden wir darüber hinaus auch Sprachkonzepte, die zu endlichen bzw. Kellerautomaten äquivalent sind, kennen lernen. Abbildung 1.1 skizziert die Struktur eines Übersetzers.

Die Aufgabenstellung ist die folgende: Gegeben sind eine „höhere" Programmiersprache QS (z.B. Pascal oder Java), die *Quellsprache*, und eine „niedere" Programmiersprache ZS, die *Zielsprache*. Gesucht ist ein Compiler C, der, gegeben ein beliebiges Programm Q in QS, ein äquivalentes Programm Z in ZS erzeugt. Z und Q sind genau dann äquivalent, wenn Z, gegeben eine formale Semantikbeschreibung von QS, für jede korrekte Eingabe die gemäß der Semantik von Q erwartete Ausgabe erzeugt. Es gibt keine Rückkopplung von späteren Phasen zu früheren Phasen. Wir werden nun Werkzeuge aus der Automatentheorie und den formalen Sprachen entwickeln, mittels

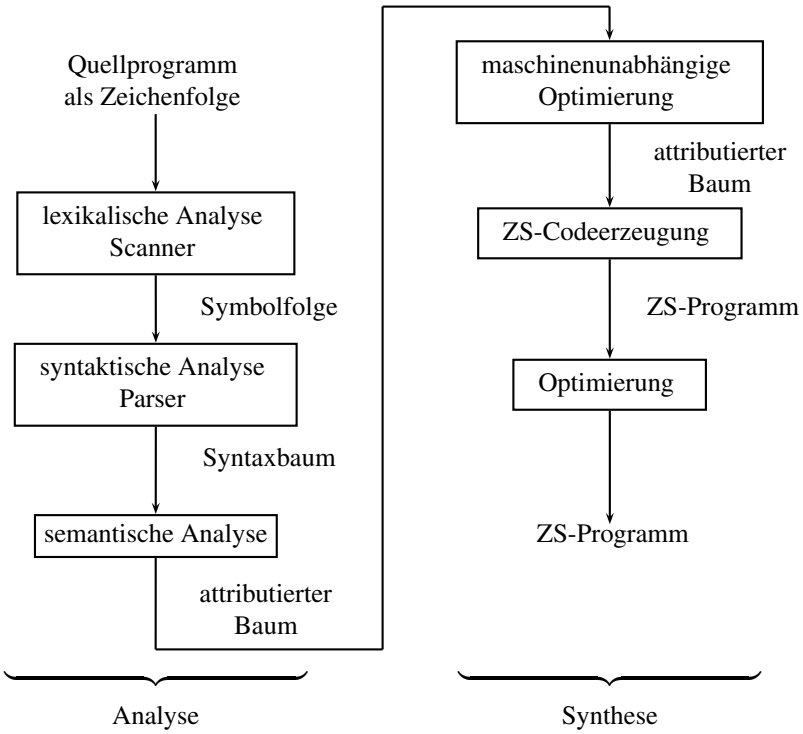


Abbildung 1.1: Struktur eines Übersetzers.

denen wir die lexikalische und die syntaktische Analyse realisieren. Die *lexikalische Analyse* dient zur Erkennung von lexikalischen Einheiten. Dabei werden reservierte Identifier wie **var**, **begin** oder **real** als Schlüsselwörter erkannt. Zusätzlich werden irrelevante lexikalische Einheiten wie Kommentare, Folgen von Leerzeichen, Zeilenvorschübe usw. eliminiert. Die *Syntaxanalyse* erkennt die syntaktische Struktur des Programms. Insbesondere wird seine syntaktische Korrektheit überprüft. Die Ausgabe der Syntaxanalyse ist, falls kein syntaktischer Fehler vorliegt, der Syntaxbaum, an dem die syntaktische Struktur des Programms leicht ablesbar ist. Wir werden uns zunächst mit der lexikalischen und dann mit der Syntaxanalyse beschäftigen.

1.1 Die lexikalische Analyse

Die lexikalische Analyse wird in einem Compiler durch den *Scanner* durchgeführt. Dieser erhält als Eingabe das Programm als Zeichenfolge. Diese wird durch den Scanner in eine Folge von lexikalischen Einheiten, den *Symbolen*, aufgeteilt. Die Menge aller möglichen Symbole einer Programmiersprache bildet eine reguläre Menge.

1.1.1 Reguläre Mengen, reguläre Ausdrücke und endliche Automaten

Ein *Alphabet* Σ ist eine endliche, nichtleere Menge von Zeichen.

Beispiel 1.1
$\Sigma_1 = \{0, 1\}$ (das so genannte binäre Alphabet),
$\Sigma_2 = \{$A, B, . . . ,Z,a,b,. . . ,z,0,. . . ,9$\}$.

$\Diamond$

Ein *Wort* oder *String* über einem Alphabet Σ ist eine endliche Folge von Zeichen aus Σ. Mit ε bezeichnen wir das *leere Wort* (Folge ohne Zeichen). Die *Länge* $|w|$ eines Wortes w ist die Anzahl der Zeichen in w.

Beispiel 1.1 (Fortführung)

$w_1 = 0010110$ ist ein Wort über Σ_1, $|w_1| = 7$.
$w_2 = \text{AB01z}$ ist ein Wort über Σ_2, $|w_2| = 5$.

⧫

Eine *reguläre Menge* über Σ wird wie folgt induktiv definiert:

1. $\emptyset$ ist eine reguläre Menge über Σ.
2. $\{\varepsilon\}$ ist eine reguläre Menge über Σ.
3. Für alle $a \in \Sigma$ ist $\{a\}$ ein reguläre Menge über Σ.
4. Seien P und Q reguläre Mengen über Σ. Dann sind auch
   a) $P \cup Q$ (*Vereinigung* von P und Q),
   b) $P \cdot Q := \{pq \mid p \in P, q \in Q\}$ (*Konkatenation* von P und Q) und
   c) $P^* := \bigcup_{n \geq 0} P^n$, wobei $P^0 := \{\varepsilon\}$, $P^n := P \cdot P^{n-1}$ für $n > 0$ (*Kleene-Abschluss* von P)

   reguläre Mengen über Σ.
5. Dies sind alle regulären Mengen über Σ.

Nach Definition ist also eine Teilmenge M von Σ^* genau dann regulär, wenn $M = \emptyset$, $M = \{\varepsilon\}$, $M = \{a\}$ für ein $a \in \Sigma$ oder M aus diesen Mengen durch eine endliche Anzahl von Anwendungen der Operationen Vereinigung, Konkatenation und Kleene-Abschluss konstruiert werden kann. Somit kann eine reguläre Menge durch einen Ausdruck beschrieben werden.

Ein *regulärer Ausdruck über* Σ und die von ihm beschriebene reguläre Menge werden wie folgt induktiv definiert:

1. $\emptyset$ ist ein regulärer Ausdruck und beschreibt die reguläre Menge $\emptyset$.

2. ε ist ein regulärer Ausdruck und beschreibt die reguläre Menge $\{\varepsilon\}$.
3. $a \in \Sigma$ ist ein regulärer Ausdruck und beschreibt die reguläre Menge $\{a\}$.
4. Seien p und q reguläre Ausdrücke, die die regulären Mengen P und Q beschreiben. Dann sind
   a) $(p|q)$ ein regulärer Ausdruck, der die reguläre Menge $P \cup Q$ beschreibt,
   b) (pq) ein regulärer Ausdruck, der die reguläre Menge $P \cdot Q$ beschreibt und
   c) (p^*) ein regulärer Ausdruck, der die reguläre Menge P^* beschreibt.
5. Dies sind alle regulären Ausdrücke über Σ.

Zur Klammereinsparung vereinbaren wir die *Prioritäten* * vor $\cdot$ vor $|$. D.h., $0|10^* = (0|(1(0^*)))$. p^+ beschreibt den regulären Ausdruck pp^*.

Beispiel 1.2 Tabelle 1.1 zeigt einige reguläre Ausdrücke und die korrespondierenden regulären Mengen.

♦

Tabelle 1.1: Beispiele für reguläre Ausdrücke und reguläre Mengen.

regulärer Ausdruck	reguläre Menge
0101	$\{0101\}$
0^*	$\{0\}^*$
$(0\|1)^*$	$\{0,1\}^*$
$a(a\|b)^*$	$\{a\}\{a,b\}^*$

Zwei reguläre Ausdrücke α und β sind *gleich* ($\alpha = \beta$), wenn sie dieselbe reguläre Menge beschreiben. Folgendes Lemma ist leicht zu beweisen:

Lemma 1.1 *Seien α, β und γ reguläre Ausdrücke. Dann gilt:*

1. $\alpha|\beta = \beta|\alpha$
2. $\emptyset^* = \varepsilon$
3. $\alpha|(\beta|\gamma) = (\alpha|\beta)|\gamma$
4. $\alpha(\beta\gamma) = (\alpha\beta)\gamma$
5. $\alpha(\beta|\gamma) = \alpha\beta|\alpha\gamma$
6. $(\alpha|\beta)\gamma = \alpha\gamma|\beta\gamma$
7. $\alpha\varepsilon = \varepsilon\alpha = \alpha$
8. $\emptyset\alpha = \alpha\emptyset = \emptyset$
9. $\alpha^* = \varepsilon|\alpha^+$
10. $(\alpha^*)^* = \alpha^*$
11. $\alpha|\alpha = \alpha$
12. $\alpha|\emptyset = \alpha$

Wir können Schreibarbeit sparen, indem wir häufig verwendeten regulären Ausdrücken Namen zuweisen, die dann bei der Definition weiterer regulärer Ausdrücke

verwendet werden können. Eine *reguläre Definition* über Σ ist eine Folge von Definitionen der Form $A_1 \rightarrow R_1, A_2 \rightarrow R_2, \ldots, A_n \rightarrow R_n$, wobei $A_1, A_2, \ldots, A_n$ paarweise verschiedene Bezeichner sind, $\{A_1, A_2, \ldots, A_n) \cap \Sigma^* = \emptyset$ und jedes R_i, $1 \leq i \leq n$, ein regulärer Ausdruck über $\Sigma \cup \{A_1, A_2, \ldots, A_{i-1}\}$ ist. Man kann in einer regulären Definition sukzessiv alle regulären Ausdrücke expandieren, indem man zunächst alle Vorkommen von A_1 durch R_1, dann alle Vorkommen von A_2 durch R_2 usw. ersetzt.

Beispiel 1.3

$$
\begin{aligned}
B &\rightarrow a|b|c|\ldots|z \\
Z &\rightarrow 0|1|2|\ldots|9 \\
ID &\rightarrow B(B|Z)^* \\
\text{IntConst} &\rightarrow ZZ^* \\
\text{RealConst} &\rightarrow \text{IntConst.IntConst}((eZZ^*)|\varepsilon)
\end{aligned}
$$

♦

Übung 1.1:

1. *Beweisen Sie Lemma 1.1.*
2. *Beweisen Sie folgende Aussagen:*
   a) *Für jede reguläre Menge gibt es einen regulären Ausdruck, der diese beschreibt.*
   b) *Jeder reguläre Ausdruck beschreibt eine reguläre Menge.*
   c) *Zu jeder regulären Menge gibt es unendlich viele reguläre Ausdrücke, die diese beschreiben.*

Reguläre Ausdrücke können Elemente der korrespondierenden regulären Menge generieren. Ein Scanner muss jedoch für einen gegebenen String x entscheiden, ob dieser Element der gegebenen regulären Menge ist. Hierzu sind endliche Automaten nützlich.

Ein endlicher Automat besteht aus einem *Eingabeband*, auf dem die Eingabe x steht, einer endlichen Steuereinheit, der *endlichen Kontrolle*, die sich stets in einem von endlich vielen Zuständen befindet und mit Hilfe eines *Lesekopfes* die Eingabe auf dem Eingabeband liest. Der Lesekopf kann nur von links nach rechts bewegt werden, so dass jedes Eingabesymbol nur einmal gelesen werden kann. Der endliche Automat kann den Zustand der endlichen Kontrolle ändern, wobei zum selben Zeitpunkt mehrere Möglichkeiten hierfür zugelassen sind. D.h., der endliche Automat ist *nichtdeterministisch*. Abbildung 1.2 skizziert einen endlichen Automaten. Formal definieren wir einen nichtdeterministischen endlichen Automaten wie folgt:

Ein *nichtdeterministischer endlicher Automat* (NEA) M ist ein 5-Tupel $M = (Q, \Sigma, \delta, q_0, F)$, wobei

1. Q eine endliche, nichtleere Menge von *Zuständen*,
2. Σ ein endliches, nichtleeres *Eingabealphabet*,
3. die *Übergangsfunktion* δ eine Abbildung von $Q \times (\Sigma \cup \{\varepsilon\})$ in die Potenzmenge von Q, d.h., $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$,
4. $q_0 \in Q$ der *Startzustand* und
5. $F \subseteq Q$ die Menge der *Endzustände*

sind.

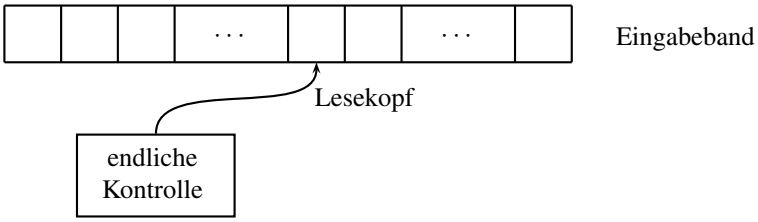


Abbildung 1.2: Endlicher Automat.

Ein Übergang, bei dem kein Zeichen der Eingabe verarbeitet wird, heißt *ε-Übergang*. Für $q \in Q$ bezeichne $FZ_\varepsilon(q)$ die Menge aller Zustände (inklusive q), die von q aus über ε-Übergänge erreichbar sind. Für $S \subseteq Q$ definieren wir $FZ_\varepsilon(S) := \bigcup_{q \in S} FZ_\varepsilon(q)$.

Wir erweitern δ zu einer Abbildung mit Definitionsbereich $2^Q \times (\Sigma \cup \{\varepsilon\})$ durch

$$\delta(\{p_1, p_2, \ldots, p_k\}, a) := \bigcup_{i=1}^{k} \delta(p_i, a). \tag{1.1}$$

Wir erweitern die Übergangsfunktion δ zu einer Funktion $\hat{\delta}$, so dass das Verhalten von M auf einem String aus Σ^* definiert durch

$$\hat{\delta}(q, \varepsilon) := FZ_\varepsilon(q) \text{ und} \tag{1.2}$$

$$\hat{\delta}(q, ax) := \bigcup_{p \in \delta(FZ_\varepsilon(q), a)} \hat{\delta}(p, x). \tag{1.3}$$

Ferner erweitern wir $\hat{\delta}$ zu einer Abbildung mit Definitionsbereich $2^Q \times \Sigma^*$ durch

$$\hat{\delta}(\{p_1, p_2, \ldots, p_k\}, x) := \bigcup_{i=1}^{k} \hat{\delta}(p_i, x). \tag{1.4}$$

Die *von M akzeptierte Sprache* $L(M)$ ist definiert durch

$$L(M) := \{x \in \Sigma^* \mid FZ_\varepsilon(\hat{\delta}(q_0, x)) \cap F \neq \emptyset\}. \tag{1.5}$$

Dies bedeutet insbesondere, dass M seine Eingabe ganz verarbeiten muss, bevor diese akzeptiert werden kann. Wir schreiben $(q, ax) \underset{M}{\vdash} (p, x)$ genau dann, wenn $p \in \hat{\delta}(q, a)$. $\overset{*}{\underset{M}{\vdash}}$ ist die reflexive, transitive Hülle von $\underset{M}{\vdash}$. Also gilt $x \in L(M)$ genau dann, wenn $(q_0, x) \overset{*}{\underset{M}{\vdash}} (q_f, \varepsilon)$ für ein $q_f \in F$. Wir sagen, M *akzeptiert* x genau dann, wenn $x \in L(M)$. Falls klar ist, welcher endliche Automat M zugrunde liegt, dann schreiben wir auch $\vdash$ bzw. $\overset{*}{\vdash}$ anstatt $\underset{M}{\vdash}$ bzw. $\overset{*}{\underset{M}{\vdash}}$.

Der aktuelle Zustand q des endlichen Automaten und die noch zu lesende Eingabe y bestimmen immer die *aktuelle Konfiguration* des Automaten. Wir schreiben diese Konfiguration als Paar (q, y). Für die Eingabe x heißt (q_0, x) *Startkonfiguration* von M bei der Eingabe x. Für $q_f \in F$ heißt (q_f, ε) *Endkonfiguration*.

Es gibt verschiedene Möglichkeiten, die Übergangsfunktion eines nichtdeterministischen endlichen Automaten darzustellen. Im Beispiel 1.4 ist die Übergangsfunktion δ mittels einer Tabelle spezifiziert.

Beispiel 1.4 Betrachten wir $M = (Q, \Sigma, \delta, q_0, F)$ mit $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_2, q_4\}$ und der in Tabelle 1.2 beschriebenen Darstellung von δ.

$\Diamond$

Tabelle 1.2: Tabellarische Darstellung von δ.

δ : Zustand	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$

Eine übersichtlichere Darstellung erhalten wir mittels eines Zustandsdiagramms. Das *Zustandsdiagramm* von M ist ein gerichteter Graph $G = (V, E)$ mit markierten Kanten, wobei $V = Q$ und für alle $a \in \Sigma \cup \{\varepsilon\}$ und alle $q, q' \in Q$ die mit $a \in \Sigma \cup \{\varepsilon\}$ markierte Kante (q, q') genau dann in E ist, wenn $q' \in \delta(q, a)$.

Wenn wir das Zustandsdiagramm zeichnen, werden Endzustände durch doppelte Kreise und der Startzustand durch $\longrightarrow$ gekennzeichnet.

Beispiel 1.4 (Fortführung) Für obigen NEA M beschreibt Abbildung 1.3 das Zustandsdiagramm. Aus dem Zustandsdiagramm ergibt sich sofort, dass die von M akzeptierte Sprache aus denjenigen Strings über $\{0,1\}$ besteht, die zwei aufeinanderfolgende Nullen oder Einsen enthalten. D.h., $L(M) = \{0,1\}^*\{00\}\{0,1\}^* \cup \{0,1\}^*\{11\}\{0,1\}^*$.

♦

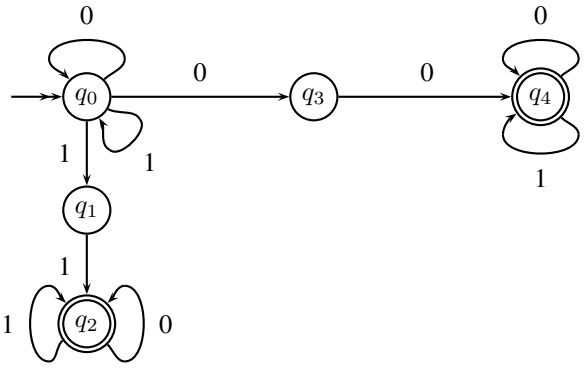


Abbildung 1.3: Zustandsdiagramm des NEA M.

Als nächstes überzeugen wir uns, dass reguläre Mengen durch nichtdeterministische endliche Automaten akzeptiert werden.

Satz 1.1 *Sei α ein regulärer Ausdruck über einem Alphabet Σ. Dann gibt es einen NEA $M(\alpha)$, der die von α beschriebene reguläre Menge akzeptiert.*

Beweis: Falls $\alpha = \emptyset$, dann genügt zur Konstruktion von $M(\alpha)$ einen endlichen Automaten zu definieren, dessen Startzustand kein Endzustand ist und der nie aus seinem Startzustand herauskommt.

Falls α den Ausdruck $\emptyset$ als Unterausdruck enthält, dann kann leicht mit Hilfe von Lemma 1.1 dieser Unterausdruck eliminiert werden. Also können wir o.B.d.A. annehmen, dass α nicht den Ausdruck $\emptyset$ als Unterausdruck enthält.

Wir geben nun einen Algorithmus an, der aus α das Zustandsdiagramm des gesuchten nichtdeterministischen endlichen Automaten $M(\alpha)$ konstruiert. Das Zustandsdiagramm für $M(\emptyset)$ besteht aus jeweils einem Knoten für den Start- und den Endzustand und keiner Kante. Für $\alpha \neq \emptyset$ starten wir mit einem Graphen, der nur den Startzustand q_0 und den Endzustand q_f als Knoten enthält. Die einzige Kante ist $q_0 \xrightarrow{\alpha} q_f$, die mit dem gesamten regulären Ausdruck α markiert ist. α wird nun gemäß seiner syntak-

tischen Struktur zerlegt, wobei gleichzeitig der aktuelle Graph gemäß der Zerlegung von α verfeinert wird. Dies wird solange fortgesetzt, bis alle Kanten im Graphen Markierungen aus $\Sigma \cup \{\varepsilon\}$ haben. Der resultierende Graph ist dann das Zustandsdiagramm von $M(\alpha)$.

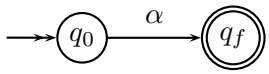
Algorithmus RA $\rightsquigarrow$ NEA:

Eingabe: regulärer Ausdruck $\alpha \neq \emptyset$ über Σ, der $\emptyset$ nicht als Unterausdruck enthält.

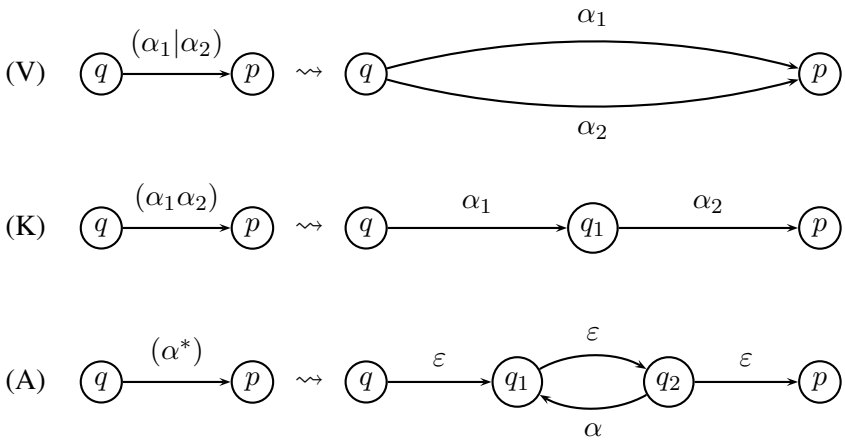
Ausgabe: Zustandsdiagramm von $M(\alpha)$.

Methode:

- Start:



- Wir wenden die nachfolgenden Regeln solange auf den jeweils aktuellen Graphen an, bis alle Kanten Markierungen aus $\Sigma \cup \{\varepsilon\}$ haben.



Die Knoten auf der linken Seite der Regel werden mit Knoten im aktuellen Graphen identifiziert. Alle in der rechten Seite einer Regel neu auftretenden Knoten entsprechen neu kreierten Knoten und somit auch neuen Zuständen.

Die Korrektheit des Algorithmus RA $\rightsquigarrow$ NEA beweisen wir mittels vollständiger Induktion über die Länge des regulären Ausdrucks α.

Falls $|\alpha| = 1$, dann hat α eine der Formen $\emptyset$, ε oder a für ein $a \in \Sigma$. Den Fall $\alpha = \emptyset$ haben wir bereits oben diskutiert und für die Eingabe vorausgesetzt, dass $\alpha \neq \emptyset$. In beiden anderen Fällen ist der Startgraph gerade das Zustandsdiagramm des gesuchten endlichen Automaten.

Nehmen wir an, dass der Algorithmus für alle α mit $|\alpha| \leq k$, $k \geq 1$ korrekt arbeitet. Sei α ein regulärer Ausdruck der Länge $k+1$. Dann hat α eine der Formen $(\beta|\gamma)$, $(\beta\gamma)$, oder (β^*), wobei die Teilausdrücke β und γ höchstens die Länge k haben. Dann folgt direkt aus der Induktionsannahme und den Zustandsdiagrammen (V), (K) und (A), dass der Algorithmus den endlichen Automaten korrekt konstruiert.

■

Da in jeder Regel maximal zwei neue Zustände kreiert werden, enthält $M(\alpha)$ höchstens $2 \cdot |\alpha|$ Zustände, wobei $|\alpha|$ die Länge von α bezeichnet. Wir werden später sehen, dass nichtdeterministische Automaten nur reguläre Mengen akzeptieren.

Reale Rechner sind keine nichtdeterministischen, sondern deterministische Maschinen. $M(\alpha)$ ist nichtdeterministisch, kann also auf einem Computer nicht direkt implementiert werden. Zunächst werden wir uns überlegen, wie wir einen nichtdeterministischen endlichen Automaten auf einem Computer simulieren können.

Oben haben wir bemerkt, dass eine Eingabe $x = x_1x_2 \ldots x_n$ genau dann von einem nichtdeterministischen endlichen Automaten M akzeptiert wird, wenn es eine Berechnung von M gibt, die x abarbeitet und dabei in einem Endzustand terminiert. Die Idee ist nun, für jeden Präfix $x_1x_2 \ldots x_i$, $0 \leq i \leq n$ von x alle Zustände zu berechnen, in denen der Automat M nach Abarbeitung von $x_1x_2 \ldots x_i$ gelangen kann. $x_1x_2 \ldots x_0$ steht für den Präfix ε von x. Bezeichne A_i die Menge der Zustände, in die M, gestartet im Startzustand q_0, nach Abarbeitung des Präfixes $x_1x_2 \ldots x_i$ gelangen kann. A_0 enthält dann den Startzustand und alle Zustände, in die M aus dem Startzustand unter alleiniger Verwendung von ε-Übergängen geraten kann. Demzufolge initialisieren wir A_0 durch

$$A_0 := FZ_\varepsilon(q_0). \tag{1.6}$$

Nehmen wir an, dass wir A_i, $i \geq 0$ berechnet haben. Unser Ziel ist nun die Berechnung von A_{i+1}. Nach A_{i+1} müssen wir alle Zustände hineinnehmen, in die wir, gestartet in einem Zustand aus A_i nach Abarbeiten des nächsten Symbols x_{i+1} gelangen können. Dies bedeutet insbesondere, dass wir nach dem Lesen des Zeichens x_i auch noch all diejenigen Zustände hinzunehmen müssen, in die M dann noch unter alleiniger Verwendung von ε-Übergängen geraten kann. Also berechnet sich A_{i+1} unter Verwendung von A_i durch

$$A_{i+1} := FZ_\varepsilon(\delta(A_i, x_{i+1})). \tag{1.7}$$

Nach der Berechnung von A_n brauchen wir nur noch zu testen, ob $A_n \cap F \neq \emptyset$ oder nicht. Die Eingabe x wird genau dann von M akzeptiert, wenn $A_n \cap F \neq \emptyset$. Die Korrektheit obiger Simulation des nichtdeterministischen endlichen Automaten M kann leicht mittels vollständiger Induktion über die Länge der Eingabe x bewiesen werden.

Betrachten wir noch einmal obige Simulation eines nichtdeterministischen endlichen Automaten. Für jedes $0 \leq i \leq n$ ist A_i ein Element der Potenzmenge der Zu-

standsmenge Q. Dies bedeutet, dass wir für A_i stets nur endlich viele Möglichkeiten haben. Dies legt die Vermutung nahe, dass wir einen zu M äquivalenten deterministischen endlichen Automaten konstruieren können. Diese enthalten maximal einen Übergang pro Zeichen des Eingabealphabets und keine ε-Übergänge. D.h., ein NEA $M = (Q, \Sigma, \delta, q_0, F)$ heißt *deterministischer endlicher Automat* (DEA), falls die Übergangsfunktion δ eine Abbildung $\delta : Q \times \Sigma \to Q$ ist.

Übung 1.2: *Beweisen Sie die Korrektheit der obigen Simulation eines nichtdeterministischen endlichen Automaten durch den Computer. Analysieren Sie die Laufzeit dieser Simulation.*

Ein deterministischer endlicher Automat kann direkt auf einem Computer implementiert werden. Unser Ziel ist, ein Verfahren zu entwickeln, das aus einem gegebenen NEA automatisch einen „guten“ DEA, der dieselbe Sprache akzeptiert, konstruiert. Zwei endliche Automaten M_1 und M_2, die dieselbe Sprache akzeptieren, heißen *äquivalent*. Zunächst werden wir zeigen, dass zu jedem NEA ein äquivalenter DEA existiert.

Satz 1.2 *Zu jedem NEA $M_1 = (Q, \Sigma, \delta, q_0, F)$ existiert ein äquivalenter DEA $M_2 = (Q', \Sigma, \delta', q_0', F')$.*

Beweis: Die Idee der Konstruktion ist ähnlich zur Simulation des nichtdeterministischen endlichen Automaten durch den Computer. Wir konstruieren einen DEA M_2, der M_1 simuliert. Dabei gilt $Q' \subseteq 2^Q$, $q_0' := FZ_\varepsilon(q_0)$, $F' := \{S \in Q' \mid S \cap F \neq \emptyset\}$ und $\delta'(S, a) := FZ_\varepsilon(\delta(S, a))$ für alle $a \in \Sigma$. Ein einzelner Zustand von M_2 ist also eine Teilmenge von Q.

Algorithmus NEA $\rightsquigarrow$ DEA:

Eingabe: NEA $M_1 = (Q, \Sigma, \delta, q_0, F)$.

Ausgabe: äquivalenter DEA $M_2 = (Q', \Sigma, \delta', q_0', F')$.

Methode:

(1) $q_0' := FZ_\varepsilon(q_0)$; $Q' := \{q_0'\}$; Markierung$(q_0') :=$ **false**;
(2) **while** $\exists S \in Q'$ mit Markierung$(S) =$ **false**
do
Wähle solches S;
Markierung$(S) :=$ **true**;
for alle $a \in \Sigma$
do
$T := FZ_\varepsilon(\delta(S, a))$;
if $T \notin Q'$
then

$Q' := Q' \cup \{T\}$;
Markierung(T) := **false**
fi;
$\delta'(S, a) := T$
od
od.

$L(M_1) = L(M_2)$ kann leicht mittels Induktion über die Länge des Eingabestrings x bewiesen werden. Hierzu genügt es zu zeigen, dass für $0 \leq i \leq |x|$ der Zustand des deterministischen Automaten nach Abarbeiten des Präfixes $x_1 x_2 \ldots x_i$ der Eingabe und die Menge A_i unserer deterministischen Simulation dieselben sind.

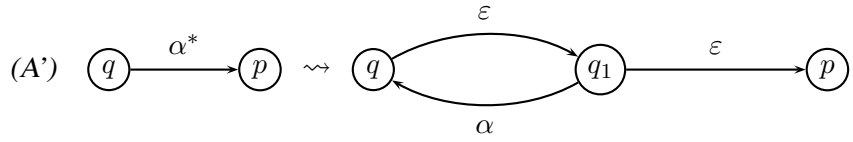
■

Übung 1.3:

a) *Sei $\alpha \neq \emptyset$ ein regulärer Ausdruck. Konstruieren Sie einen zu α äquivalenten Ausdruck α', der nicht $\emptyset$ als Unterausdruck enthält.*

b) *Beweisen Sie die Korrektheit des Algorithmus RA $\rightsquigarrow$ NEA.*

c) *Die einzige Regel, die zwei neue Zustände kreiert, ist die Regel (A). Können wir diese Regel durch folgende Regel (A') ersetzen?*



Begründen Sie Ihre Antwort.

d) *Vervollständigen Sie den Beweis des Satzes 1.2. D.h., zeigen Sie, dass $L(M_1) = L(M_2)$.*

Wir zeigen nun, dass nichtdeterministische und deterministische Automaten nur reguläre Mengen akzeptieren. Da zu jedem NEA ein äquivalenter DEA existiert, genügt es zu zeigen, dass die von einem DEA akzeptierte Menge regulär ist.

Satz 1.3 *Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DEA. Dann ist $L(M)$ regulär.*

Beweis: Sei $Q = \{q_0, q_1, \ldots, q_n\}$. Es genügt, einen regulären Ausdruck α zu konstruieren, der die Menge $L(M)$ beschreibt. Für $i, j \in \{0, 1, \ldots, n\}$ und $k \in \{-1, 0, 1, \ldots, n\}$ sei R_{ij}^k die Menge aller Worte über Σ, für die M, gestartet im Zustand q_i, den DEA in den Zustand q_j überführt, ohne zwischendurch einen Zustand q_l mit $l > k$ anzunehmen. Dabei sind $i > k$ und $j > k$ erlaubt. Wir können R_{ij}^k wie

folgt induktiv definieren:

$$R_{ij}^{-1} := \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & \text{falls } i \neq j \\ \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & \text{falls } i = j. \end{cases} \tag{1.8}$$

In beiden Fällen ist R_{ij}^{-1} endlich und kann leicht durch einen regulären Ausdruck beschrieben werden. Sei α_{ij}^{-1} ein regulärer Ausdruck für R_{ij}^{-1}.

Sei $k > -1$. Jeder String $x \in R_{ij}^{k} \setminus R_{ij}^{k-1}$ kann geschrieben werden als $x = vwy$, wobei

1. M, gestartet mit Eingabe v im Zustand q_i, in den Zustand q_k übergeht, ohne zwischendurch einen Zustand q_l mit $l > k-1$ anzunehmen (d.h., $v \in R_{ik}^{k-1}$),
2. M, gestartet mit Eingabe w im Zustand q_k, in den Zustand q_k übergeht, ohne zwischendurch einen Zustand q_l mit $l > k$ anzunehmen (d.h., $w \in \left(R_{kk}^{k-1}\right)^*$) und
3. M, gestartet mit Eingabe y im Zustand q_k, in den Zustand q_j übergeht, ohne zwischendurch einen Zustand q_l mit $l > k-1$ anzunehmen (d.h., $y \in R_{kj}^{k-1}$).

Also erhalten wir

$$R_{ij}^{k} := R_{ij}^{k-1} \cup R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1}. \tag{1.9}$$

Seien $\alpha_{ij}^{k-1}, \alpha_{ik}^{k-1}, \alpha_{kk}^{k-1}$ und α_{kj}^{k-1} reguläre Ausdrücke für $R_{ij}^{k-1}, R_{ik}^{k-1}, R_{kk}^{k-1}$ und R_{kj}^{k-1}. Dann ist

$$\alpha_{ij}^{k} := \alpha_{ij}^{k-1} \mid \alpha_{ik}^{k-1} \left(\alpha_{kk}^{k-1}\right)^* \alpha_{kj}^{k-1} \tag{1.10}$$

ein regulärer Ausdruck für R_{ij}^{k}.

Sei $F = \{q_{j_1}, q_{j_2}, \ldots, q_{j_t}\}$ die Menge der Endzustände des endlichen Automaten M. Dann gilt

$$L(M) = \bigcup_{q_{j_i} \in F} R_{0j_i}^{n}. \tag{1.11}$$

Also ist

$$\alpha = \alpha_{0j_1}^{n} | \ldots | \alpha_{0j_t}^{n} \tag{1.12}$$

ein regulärer Ausdruck für $L(M)$.

■

Satz 1.3 wird zwar für die Konstruktion eines Scanners nicht benötigt, bildet jedoch eine schöne Abrundung für die oben entwickelte Theorie der regulären Mengen. Wenden wir uns wieder der Konstruktion eines Scanners zu. Der durch den Algorithmus

NEA $\rightsquigarrow$ DEA konstruierte DEA M_2 ist nicht notwendigerweise ein „guter“ DEA. D.h., es könnte ein äquivalenter DEA mit wesentlich weniger Zuständen existieren. In diesem Fall wäre es ungeschickt, M_2 direkt zur Konstruktion des Scanners zu verwenden.

Ein DEA M heißt *minimal*, falls jeder DEA M' mit $L(M') = L(M)$ mindestens soviele Zustände wie M enthält. Unser Ziel ist ein Verfahren zu entwickeln, das aus einem gegebenen DEA M einen minimalen DEA M' mit $L(M') = L(M)$ konstruiert.

1.1.2 Minimierung endlicher Automaten

Zunächst werden wir für eine gegebene reguläre Menge den optimalen DEA genau charakterisieren. Hierfür benötigen wir folgende Bezeichnung:

Der *Index* einer Äquivalenzrelation ist die Anzahl ihrer Äquivalenzklassen. Eine Äquivalenzrelation $R \subseteq \Sigma^* \times \Sigma^*$ heißt *rechtsinvariant*, falls $a\,R\,b \Rightarrow \forall z \in \Sigma^* : az\,R\,bz$.

Satz 1.4 *Folgende drei Aussagen sind äquivalent:*

1. *Die Menge $L \subseteq \Sigma^*$ wird durch einen DEA akzeptiert.*
2. *L ist die Vereinigung einiger Äquivalenzklassen einer rechtsinvarianten Äquivalenzrelation von endlichem Index.*
3. *Sei R_L definiert durch $x\,R_L\,y \Leftrightarrow \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L$. Dann hat die Äquivalenzrelation R_L einen endlichen Index.*

Beweis:
$1 \Rightarrow 2$:
Nehmen wir an, dass $L = L(M)$ für einen DEA $M = (Q, \Sigma, \delta, q_0, F)$. Sei die Äquivalenzrelation R_M definiert durch $x\,R_M\,y \Leftrightarrow \delta(q_0, x) = \delta(q_0, y)$. D.h., zwei Strings x und y sind genau dann in derselben Äquivalenzklasse, wenn M, gestartet im Startzustand q_0, bei Eingabe x und bei Eingabe y in denselben Zustand gelangt. Für alle $z \in \Sigma^*$ gilt $\delta(q_0, x) = \delta(q_0, y) \Rightarrow \delta(q_0, xz) = \delta(q_0, yz)$. Also ist R_M rechtsinvariant. Ferner ist der Index von $R_M \leq |Q|$ und somit endlich. Aus der Definition von R_M ergibt sich direkt, dass L die Vereinigung derjenigen Äquivalenzklassen ist, die ein x mit $\delta(q_0, x) \in F$ enthalten.

$2 \Rightarrow 3$:
Sei R eine Äquivalenzrelation, die die Aussage 2 erfüllt. Wir zeigen, dass jede Äquivalenzklasse von R in einer Äquivalenzklasse von R_L enthalten ist, woraus direkt $\text{Index}(R_L) \leq \text{Index}(R)$ folgt. Da R endlichen Index hat, ist somit der Index von R_L auch endlich.

Betrachten wir $x, y \in \Sigma^*$ mit $x\,R\,y$. Da R rechtsinvariant ist, gilt $xz\,R\,yz$ für alle $z \in \Sigma^*$. L ist eine Vereinigung von Äquivalenzklassen von R. Daraus folgt, da xz und

yz in derselben Äquivalenzklasse von R sind, $xz \in L \Leftrightarrow yz \in L$. Also gilt $x\, R_L\, y$ und somit auch $[x]_R \subseteq [x]_{R_L}$, wobei $[x]_R$ diejenige Äquivalenzklasse von R bezeichnet, die x enthält. Wenn klar ist, welche Äquivalenzrelation R gemeint ist, dann schreiben wir auch $[x]$ anstatt $[x]_R$.

$3 \Rightarrow 1$:
Aus der Definition von R_L ist leicht herzuleiten, dass R_L rechtsinvariant ist. Wir werden nun einen DEA $M' = (Q', \Sigma, \delta', q_0', F')$ mit $L(M') = L$ definieren. Hierzu seien

$$\begin{aligned} Q' &:= \text{ Menge der Äquivalenzklassen von } R_L, \\ \delta'([x], a) &:= [xa] \text{ für alle } x \in \Sigma^*, a \in \Sigma, \\ q_0' &:= [\varepsilon] \text{ und} \\ F' &:= \{[x] \mid x \in L\}. \end{aligned}$$

Da R_L rechtsinvariant ist, ist die Definition von δ' konsistent. Wegen $\delta'(q_0, x) = [x]$ und $x \in L(M') \Leftrightarrow [x] \in F' \Leftrightarrow x \in L$ gilt $L(M') = L$.

■

Im Beweis des Satzes 1.4 haben wir für einen gegebenen DEA $M = (Q, \Sigma, \delta, q_0, F)$ eine Äquivalenzrelation R_M konstruiert, für die

1. Index$(R_M) \leq |Q|$ und
2. $L(M)$ die Vereinigung einiger Äquivalenzklassen von R_M sind.

Dann haben wir gezeigt, dass Index$(R_M) \geq$ Index$(R_{L(M)})$. Schließlich haben wir mit Hilfe von $R_{L(M)}$ einen DEA $M' = (Q', \Sigma, \delta', q_0', F')$ mit

1. $L(M') = L(M)$ und
2. $|Q'| =$ Index$(R_{L(M)})$

konstruiert. Da insbesondere R_M immer eine Verfeinerung der Äquivalenzrelation $R_{L(M)}$ ist, ergibt sich direkt folgendes Korollar:

Korollar 1.1 *Sei $L \subseteq \Sigma^*$ eine reguläre Menge. Dann ist der DEA mit minimaler Anzahl von Zuständen, der L akzeptiert, bis auf Isomorphie (d.h. Umbenennung der Zustände) eindeutig bestimmt und durch M' im Beweis des Satzes 1.4 gegeben.*

Sei ein beliebiger DEA $M = (Q, \Sigma, \delta, q_0, F)$ gegeben. Betrachten wir einen beliebigen Zustand $q \in Q$. Wir sagen q ist *erreichbar*, falls ein $x \in \Sigma^*$ mit $\delta(q_0, x) = q$ existiert. Andernfalls heißt q *unerreichbar*. Wir können unerreichbare Zustände aus Q streichen, ohne dadurch $L(M)$ zu verändern. Wir sagen, M ist *reduziert*, wenn Q keine unerreichbaren Zustände enthält. O.B.d.A. sei M reduziert. Dann gibt es eine Bijektion zwischen den Zuständen in Q und den Äquivalenzklassen von R_M.

Übung 1.4: *Entwickeln Sie einen Algorithmus, der alle unerreichbaren Zustände aus Q in $O(|\Sigma| \cdot |Q|)$ Zeit streicht.*

Unsere Zielsetzung ist es nun, diejenigen Zustände zusammenzufassen, deren korrespondierende Äquivalenzklassen von R_M in derselben Äquivalenzklasse von $R_{L(M)}$ enthalten sind. Wir vereinbaren $\delta(q, \varepsilon) := q$ für alle $q \in Q$. Aus der Definition von $R_{L(M)}$ ergibt sich sofort, dass wir zwei Zustände $p, q \in Q$ genau dann zusammenfassen möchten, wenn für alle $z \in \Sigma^*$ der DEA M, gestartet im Zustand p, die Eingabe z genau dann akzeptiert, wenn M, gestartet im Zustand q, die Eingabe z akzeptiert. Also ist unsere Zielsetzung, die Äquivalenzklassen folgender Äquivalenzrelation $\equiv$ über Q zu berechnen:

Zwei Zustände $p, q \in Q$ heißen genau dann *äquivalent*, wenn für alle $z \in \Sigma^*$ gilt: $\delta(p, z) \in F \Leftrightarrow \delta(q, z) \in F$. Wir schreiben dann $p \equiv q$. Also gilt: $p \not\equiv q \Leftrightarrow \exists x \in \Sigma^* : \delta(p, x) \in F$ und $\delta(q, x) \notin F$ oder umgekehrt. Falls $p \not\equiv q$, dann sagen wir, dass p und q *unterscheidbar* sind. Anstatt mit den einzelnen Zuständen von Q zu starten und diese dann zu den Äquivalenzklassen von $\equiv$ zusammenzufassen könnte man auch mit der Vergröberung F und $Q \setminus F$ dieser Äquivalenzklassen starten und diese sukzessive verfeinern, bis die Äquivalenzklassen von $\equiv$ berechnet sind. Hierzu ist folgendes Lemma nützlich:

Lemma 1.2 $p \not\equiv q \Leftrightarrow \exists a \in \Sigma \cup \{\varepsilon\} : \delta(p, a) \not\equiv \delta(q, a)$.

Beweis: Falls $\delta(p, a) \not\equiv \delta(q, a)$ für ein $a \in \Sigma \cup \{\varepsilon\}$, dann folgt direkt aus der Definition der Äquivalenzrelation $\equiv$, dass $p \not\equiv q$.

Nehmen wir an, dass $p \not\equiv q$. Gemäß der Definition der Relation $\equiv$ existiert dann ein String $x \in \Sigma^*$ mit $\delta(p, x) \in F$ und $\delta(q, x) \notin F$ oder umgekehrt. Sei $x \in \Sigma^*$ ein kürzester solcher String.

Falls $x = \varepsilon$, dann ist genau einer der beiden Zustände in F, woraus die Behauptung folgt. Falls $x \neq \varepsilon$, dann gilt $x = ay$ für ein $a \in \Sigma$, $y \in \Sigma^*$. Betrachten wir $\delta(p, a)$ und $\delta(q, a)$. Aus der Konstruktion folgt, dass $\delta(p, a)$ und $\delta(q, a)$ durch y unterschieden werden. Also gilt $\delta(p, a) \not\equiv \delta(q, a)$.

■

Falls die Übergangsfunktion δ nicht total ist, dann können wir dies durch Hinzunahme eines neuen Zustandes r erreichen. Wir erweitern die Übergangsfunktion δ für alle $q \in Q$ und $a \in \Sigma$ wie folgt:

$$\delta(q, a) = r \text{ falls } \delta(q, a) \text{ undefiniert und} \tag{1.13}$$

$$\delta(r, a) = r \text{ für alle } a \in \Sigma \cup \{\varepsilon)\}. \tag{1.14}$$

Lemma 1.2 impliziert, dass Zustände in F und Zustände in $Q \setminus F$ in verschiedenen Äquivalenzklassen von $\equiv$ liegen müssen. Also bilden die beiden Mengen F und $Q \setminus F$ eine Vergröberung der Äquivalenzklassen von $\equiv$. Wir starten nun mit dieser Vergröberung und verfeinern diese sukzessive, bis die Äquivalenzklassen von $\equiv$ berechnet sind. Es folgt unmittelbar, dass für eine Menge K von Zuständen in der aktuellen Verfeinerung stets $K \subseteq F$ oder $K \subseteq Q \setminus F$ gilt. Somit erhalten wir folgenden Algorithmus:

Algorithmus DEA $\rightsquigarrow$ DEA$_{min}$

Eingabe: reduzierter DEA $M = (Q, \Sigma, \delta, q_0, F)$ mit δ total.

Ausgabe: äquivalenter minimaler DEA $M' = (Q', \Sigma, \delta', q_0{}', F')$.

Methode:

(1) $t := 2$; $Q_1 := F$; $Q_2 := Q \setminus F$.

(2) **while** $\exists i \leq t$, $a \in \Sigma$ mit $\delta(Q_i, a) \not\subseteq Q_j$ für alle $j \leq t$
do
1. Wähle solch ein $i \leq t$, $a \in \Sigma$ und $j \leq t$
mit $\delta(Q_i, a) \cap Q_j \neq \emptyset$.
2. $Q_{t+1} := \{q \in Q_i \mid \delta(q, a) \in Q_j\}$;
$Q_i := Q_i \setminus Q_{t+1}$;
$t := t + 1$

od.

(3) $Q' := \{Q_1, Q_2, \ldots, Q_t\}$;
$q_0' := [q_0]$;
$F' := \{[q] \in Q' \mid q \in F\}$;
$\delta'([q], a) := [\delta(q, a)]$ für alle $q \in Q$, $a \in \Sigma$.

Dabei bezeichnet $[q]$ für $q \in Q$ diejenige Klasse Q_j in Q', die q enthält.

Die Korrektheit des Algorithmus ergibt sich unmittelbar aus Lemma 1.2.

Übung 1.5:

a) Beweisen Sie die Korrektheit des Algorithmus DEA $\rightsquigarrow$ DEA$_{min}$.

b) Seien $|\Sigma| = k$ und $|Q| = n$. Zeigen Sie, dass obiger Algorithmus derart implementiert werden kann, dass seine Laufzeit $O(kn^2)$ ist.

Anhand des obigen Algorithmus kann schön demonstriert werden, dass ein wenig Nachdenken, ob mit grundlegenden Datenstrukturen und Standardmethoden eine effizientere Implementierung möglich ist, sich lohnen kann.

Schritt (2) ist der einzige Teil des Algorithmus, der nicht leicht in linearer Zeit implementiert werden kann. Die naheliegende Implementierung der Überprüfung der

Bedingung der **while**-Schleife sowie die Ausführung des Blockes der **while**-Schleife ist derart aufwendig, dass im worst case hierfür insgesamt $O(kn^2)$ Zeit benötigt wird. Falls wir für alle $q \in Q$ garantieren könnten, dass sich der Index der Menge, die q enthält, maximal $\log n$-mal ändert, dann sind wir einer $O(kn \log n)$-Implementierung schon wesentlich näher. Hierzu modifizieren wir den Block der **while**-Schleife wie folgt:

1. Wähle solch ein $i \leq t$, $a \in \Sigma$ und $j_1, j_2 \leq t$
   mit $j_1 \neq j_2$, $\delta(Q_i, a) \cap Q_{j_1} \neq \emptyset$ und $\delta(Q_i, a) \cap Q_{j_2} \neq \emptyset$.
2. **if** $|\{q \in Q_i \mid \delta(q, a) \in Q_{j_1}\}| \leq |\{q \in Q_i \mid \delta(q, a) \in Q_{j_2}\}|$
   then
   $Q_{t+1} := \{q \in Q_i \mid \delta(q, a) \in Q_{j_1}\}$
   else
   $Q_{t+1} := \{q \in Q_i \mid \delta(q, a) \in Q_{j_2}\}$
   fi;
   $Q_i := Q_i \setminus Q_{t+1}$;
   $t := t + 1$.

Aus der Wahl von i folgt direkt, dass j_1 und j_2 existieren. Ferner gilt $|Q_{t+1}| \leq \frac{1}{2}|Q_i|$, bezüglich $|Q_i|$ vor der Neudefinition von Q_i. Falls sich für einen Zustand q der Index derjenigen Menge, die q enthält, ändert, dann halbiert sich zumindest die Größe der korrespondierenden Menge. Also kann sich für alle $q \in Q$ der Index derjenigen Menge, die q enthält, maximal $\log n$-mal ändern.

Unser Ziel ist es nun, eine Implementierung zu entwickeln, so dass die Gesamtarbeit Transitionen zugerechnet werden kann, die einen Zustand enthalten, für den sich der Index der korrespondierenden Klasse ändert. Hierfür benötigen wir eine Datenstruktur, die folgende Operationen unterstützt:

1. Die Wahl von $i \leq t$, $a \in \Sigma$ mit $\delta(Q_i, a) \not\subseteq Q_j$, für alle $j \leq t$;
2. die Wahl von $j_1, j_2 \leq t$, $j_1 \neq j_2$ mit $\delta(Q_i, a) \cap Q_{j_1} \neq \emptyset$ und $\delta(Q_i, a) \cap Q_{j_2} \neq \emptyset$;
3. die Entscheidung ob $|\{q \in Q_i \mid \delta(q, a) \in Q_{j_1}\}| \leq |\{q \in Q_i \mid \delta(q, a) \in Q_{j_2}\}|$ und
4. die Konstruktion von Q_{t+1} und $Q_i \setminus Q_{t+1}$.

Abbildung 1.4 illustriert die nachfolgend beschriebene Datenstruktur.

Bezeichne D_t diejenige Datenstruktur, die wir direkt nach der Konstruktion von Q_t, $t \geq 2$, erhalten haben. Im wesentlichen besteht D_t aus folgender Vergröberung δ' von δ:

$$\delta' \subseteq \{1, 2, \ldots, t\} \times \Sigma \times \{1, 2, \ldots, t\}, \text{ wobei}$$
$$(i, a, j) \in \delta' \Leftrightarrow \exists q \in Q_i, p \in Q_j \text{ mit } \delta(q, a) = p.$$

Für jedes Tripel $(i,a,j) \in \delta'$ enthält D_t eine doppelt verkettete Liste $L(i,a,j)$, die genau diejenigen Zustände $q \in Q_i$ mit $\delta(q,a) \in Q_j$ enthält. Jedes Element der Liste hat einen zusätzlichen Zeiger auf den Kopf der Liste, der alle benötigten Informationen enthält. Die Größe der Liste $L(i,a,j)$ ist in der Variablen $S(i,a,j)$ gespeichert. Also kann die Größe der Liste $L(i,a,j)$ in konstanter Zeit ermittelt werden. Zusätzlich enthält die Datenstruktur ein $(|Q| \times |\Sigma|)$-Feld Δ und ein $(|\Sigma| \times |Q|)$-Feld Δ^{-1}. Die Komponente $\Delta(q,a)$, $q \in Q$, $a \in \Sigma$, enthält einen Zeiger auf die bezüglich den Listen $L(\cdot,a,\cdot)$ eindeutig bestimmte Feldkomponente, die q enthält. Die Eindeutigkeit dieser Feldkomponente ergibt sich direkt daraus, dass der zugrundeliegende endliche Automat deterministisch ist. Die Komponente $\Delta^{-1}(b,p)$, $b \in \Sigma$, $p \in Q$ enthält einen Zeiger auf eine Liste, die exakt diejenigen Zustände $q \in Q$ mit $\delta(q,b) = p$ enthält. Wir identifizieren diese Menge von Zuständen mit $\Delta^{-1}(b,p)$. Für jedes Paar (i,a), $i \in \{1,2,\ldots,t\}$, $a \in \Sigma$ verwalten wir eine Liste $\Delta'(i,a)$, die genau dann einen Zeiger auf den Listenkopf von $L(i,a,j)$ enthält, wenn $(i,a,j) \in \delta'$. Der Listenkopf von $L(i,a,j)$ enthält dann einen Zeiger auf diejenige Komponente in $\Delta'(i,a)$, die den Zeiger auf $L(i,a,j)$ enthält. In einer Menge K verwalten wir Zeiger auf diejenigen Listen $\Delta'(i,a)$, deren Länge ≥ 2 ist. Der Kopf der Liste $\Delta'(i,a)$ enthält einen Zeiger auf dasjenige Element von K, das auf $\Delta'(i,a)$ zeigt. Zusätzlich zu D_t benötigen wir zur effizienten Berechnung von D_{t+1} zwei weitere Felder Γ und Γ'. Deren Struktur und Funktion ergibt sich aus der unten beschriebenen Konstruktion der Datenstruktur D_{t+1}.

D_2 kann leicht in $O(kn)$ Zeit konstruiert werden.

Übung 1.6: *Zeigen Sie, dass die Datenstruktur D_2 in $O(kn)$ Zeit aufgebaut werden kann.*

Nehmen wir an, dass die Datenstruktur D_t, $t \geq 2$ gegeben ist. Wir beschreiben nun die Durchführung des Schrittes (2) bezüglich der Konstruktion von D_{t+1}.

1. Unter Verwendung der Menge K bestimmen wir in konstanter Zeit eine Liste $\Delta'(i,a)$ der Länge ≥ 2 und wählen zwei beliebige Tripel (i,a,j_1), (i,a,j_2) in $\Delta'(i,a)$ aus. Mit Hilfe von $S(i,a,j_1)$ und $S(i,a,j_2)$ entscheiden wir, ob $|L(i,a,j_1)| \leq |L(i,a,j_2)|$. Bezeichne $L(i,a,j_{\min})$ die kürzere Liste. Nun ist die alte Liste $L(i,a,j_{\min})$ die neue Liste $L(t+1,a,j_{\min})$ und daher auch $S(t+1,a,j_{\min}) = S(i,a,j_{\min})$. Wir streichen nun den Zeiger auf $L(i,a,j_{\min})$ aus $\Delta'(i,a)$ und fügen einen Zeiger auf $L(t+1,a,j_{\min})$ in eine neue Liste $\Delta'(t+1,a)$ ein. Falls die Länge der Liste $\Delta'(i,a)$ nun kleiner als zwei ist, dann streichen wir den Zeiger auf $\Delta'(i,a)$ aus K.
2. Da sich für $q \in L(t+1,a,j_{\min})$ der Index seiner Menge von i nach $t+1$ geändert hat, müssen wir für jedes $b \in \Sigma \setminus \{a\}$ die zu q korrespondierende Komponente aus seiner Liste $L(i,b,k)$ entfernen und in die Liste $L(t+1,b,k)$