# Vorlesungszyklus
# „Formale Grundlagen der Informatik“

# F2 – Automaten und Formale Sprachen

Prüfungsunterlagen zur Vorlesung
im Sommersemester 2002
Dr. Berndt Farwer



(Überarbeitete Fassung mit Ergänzungen von Berndt Farwer)
gedruckt: 2. April 2002

**Anmerkungen zur vorliegenden Ausgabe:**

Das Skript basiert auf den F1-Skripten der Wintersemester 1998 und 1999. Nach den Erfahrungen der ersten Veranstaltungen im F-Zyklus der neuen Studienordnung, wurden mit der Platzierung dieser Veranstaltung im zweiten Semester des Grundstudiums, einige Inhalte anders dargestellt, umgeordnet und einige Teile gekürzt oder sogar weggelassen. Entsprechend eines Beschlusses des Fachbereichsrates (FBR) wurde auch der bisherige Titel „Automaten und Kalküle“ zu „Automaten und Formale Sprachen“ geändert. Durch diese Maßnahmen wird diese Veranstaltung inhaltlich besser auf die anderen Veranstaltungen des Grundstudiums abgestimmt.

Mit Unterstützung von Übungsgruppenleiter(inne)n und Besuchern früherer F1- bzw. F2-Veranstaltungen konnten (einige) Fehler in dieser Ausgabe vermieden werden. Sie halten eine korrigierte und ergänzte Version in den Händen. Dennoch sind Fehler nie ganz auszuschließen. Wenn Sie meinen, einen Fehler gefunden zu haben, dann wenden Sie sich bitte an Prof. Dr. Matthias Jantzen oder an Dr. Berndt Farwer.

Speziell danken möchten die Autoren jenen Studierenden, die sich in den Übungsgruppen und der Vorlesung fragend aber auch kritisch zu Wort meldeten! Allen, die zur Fehlerfreiheit und Verbesserung des Skriptes beigetragen haben, gebührt besonderer Dank: Alex Bepple, Jan Ortmann, Wilfried Röper, Christina Theilmann und Matthias Wester–Ebbinghaus (Namen in alphabetischer Reihenfolge) sowie viele hier nicht namentlich genannte Studierende halfen durch ihre Korrekturen, Unsauberkeiten und Fehler vorangegangener Ausgaben zu korrigieren!

Das vorliegende Skript wurde mit LaTeX2e unter Verwendung diverser Zusatzpakete auf einem Power-Macintosh unter MacOS X erstellt. Die Zeichnungen wurden zum größten Teil mit den Programmen OmniGraffle, Illustrator v. 8.0 bis 10.0 (von Adobe) und Canvas v. 5.0 u. 6.0 (von Deneba) als EPS-Dokumente (*encapsulated postscript files*) erstellt.

# Inhaltsverzeichnis

# 1 Einleitung

„Die Grenzen meiner Welt sind die Grenzen meiner Sprache.“
„Ich kann in einer Disziplin nur soviel an Wissenschaft entdecken, wie Mathematik in ihr enthalten ist.“
(Ludwig Wittgenstein, 1889 – 1951, Philosoph und Mathematiker)

Informatik ist eine Wissenschaft, die sich mit der Beschreibung, Analyse und Gestaltung informationsverarbeitender Prozesse beschäftigt. Zur Analyse und Aufbereitung von Problemen aus dem täglichen Leben wie auch der Wissenschaft, ist es nötig, ihre strukturellen und mathematischen Aspekte zu erkennen und zu formulieren. Oft erkennt man diese erst, wenn auch die Möglichkeiten zu deren Formulierung zur Verfügung stehen. Da eine Hauptaufgabe der Informatik die Entwicklung von maschinell durchführbaren Verfahren zur Lösung von Problemen der Informationsverarbeitung ist, kommt der formalen Beschreibung der Strukturen, Modelle und Algorithmen eine besondere Bedeutung zu. Während jedoch die Mathematik wie selbstverständlich mit unendlichen Objekten, wie zum Beispiel den reellen Zahlen, oder Zahlen, die nicht rational sind wie $\pi$ oder $e$, aber auch mit unendlichen Mengen umgeht, diese definiert, umrechnet und darüber Beweise führt, ist das in der Informatik manchmal nur sehr eingeschränkt möglich. Hier benötigen wir reale Rechner, die keine wirklich unendlichen Objekte bearbeiten können. An deren Stelle werden immer endliche Repräsentationen treten müssen, mit denen dann weiter gearbeitet werden kann. In der (nicht nur theoretischen) Informatik, haben wir es zum größten Teil mit Fragen zu endlichen Repräsentationen beliebiger Objekte und deren adäquaten algorithmischen Behandlung zu tun.

Programme zur Ausführung von Algorithmen auf Rechenmaschinen sind nun kaum brauchbar, wenn sie das gestellte Problem nicht korrekt lösen; oder dies zwar richtig bearbeiten, jedoch nicht mit der nötigen Effizienz. Aussagen über die Korrektheit und Effizienz von Programmen sind aber nur möglich, wenn Syntax und Semantik der verwendeten Konstruktionen und Programmiersprachen exakt festgelegt sind. Die Methoden, die solches erst möglich machen, sind Gegenstand der Theoretischen Informatik. Gleichzeitig mit der formal einwandfreien, mathematischen Formulierung von Modellen, wird der Rahmen geliefert, in dem versprochene Eigenschaften bewiesen werden können.

Die **Theoretische Informatik** befaßt sich mit formalen Grundlagen. In dem Teilgebiet **Automatentheorie** wird zum Beispiel danach gefragt, welche einfachen mathematischen Modelle dem Computer zu Grunde liegen. Die **Theorie der Berechenbarkeit** untersucht, wie man das Berechenbare von dem Nichtberechenbaren abgrenzen kann, indem man Probleme benennt, die ein Computer unter keinen Umständen lösen kann. Die **Komplexitätstheorie** fragt danach, welchen rechnerischen Aufwand die Lösung gewisser Problem erfordert. In dem Teilgebiet **Formale Sprachen** wird der prinzipielle, strukturelle Aufbau von Programmier- und Spezifikationssprachen (deren **Syntax**) untersucht. Die **Logik** bildet die Grundlage für eine formale Semantik von Konstruktionen z.B. mit Programmier- und Spezifikationssprachen.

Die Mathematik spielt in der Informatik die Rolle eines „Denkzeugs“, mit dem wir knapp und präzise, ohne Vagheiten und Mehrdeutigkeiten den (abstrakten) Kern einer Sache beschreiben können. Erst auf der Basis eines sauberen theoretischen Fundaments wird es möglich, solche Beschreibungen zu formulieren und deren Analysen vorzunehmen.

Zum Verständnis der (nicht nur theoretischen) Grundlagen der Informatik wird im Anhang, quasi als Zusammenschau und Überblick, ein kurzer Einblick in die historischen Ursprünge dargestellt. Die Autoren beschränken sich hier jedoch auf den Teil, der dem Vorlesungszyklus **Formale Grundlagen der Informatik** am nächsten steht. Veranstaltungen der übrigen Grundstudiumszyklen, hier vor allem der IMG-Zyklus, werden weitere nützliche Informationen zu den Grundlagen der Informatik liefern.

Das vorliegende Skript zur Vorlesung „Automaten und Formale Sprachen (F2)“ des Informatik Grundstudiums soll die wesentlichen Notationen und Ergebnisse dieses grundlegenden Teils der Theoretischen Informatik wiedergeben, wobei zur Motivation Anwendungen angesprochen werden. Die zum Verständnis notwendigen mathematischen Notationen und Konzepte werden angesprochen, dürften aber zum großen Teil auch in den vorangegangenen Mathematik-Veranstaltungen bekannt gemacht worden sein und können entsprechenden Lehrbüchern entnommen werden. Trotzdem werden einzelne Beweise relativ ausfürlich aufgenommen, weil dadurch die Leserinnen und Leser weitere Anwendungen der in der Mathematik vorgestellten Beweistechniken vorfinden und mit diesen vertraut gemacht werden sollen. Die dabei auftauchende formale Strenge ist jedoch keine Feind der Einfachheit. Schon Hilbert wies darauf hin, dass die Benutzung streng formaler Methoden dazu anhält nach klareren und einfacheren Beweisen zu suchen!

Am Ende dieser Unterlagen werden weitere Quellen angegeben, die zum Nachschlagen wie zur Vertiefung des Verständnisses herangezogen werden können.

Zur Einübung sind Übungsaufgaben in den angebotenen Übungsgruppen zu lösen. Diese werden entsprechend ihres unterschiedlichen Schwierigkeitsgrades mit Punkten bewertet. Die aktive Teilnahme an den angebotenen Übungsgruppen ist erforderlich. Es wird nachdrücklich empfohlen, den Stoff in kleinen Arbeitsgruppen von zwei bis höchstens vier Studierenden zu bearbeiten. Besser als durch eigenes Auseinandersetzen mit dem Stoff und zusätzliche Diskussion mit Kommilitonen kann sich ein(e) Studierende[r] kaum für eine Prüfung vorbereiten!

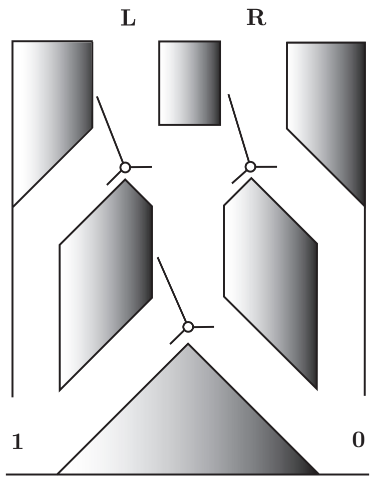
Wem eine Schreibweise, wie z.B. die vorstehende, nicht klar ist, soll hier, genauso wie später im Text, keine eindeutige Definition vermissen müssen. Damit also schon zu Beginn klar wird, wie diese Schreibweise zu verstehen ist, hier eine erste, ganz fachfremde „Definition“:

## 1.1 Schreibweise

In diesem Text benutzen wir in der Regel die Bezeichnung „Studierende“ für weibliche wie männliche „StudentInnen“. Letztere Schreibweise mit dem großen I, anstelle etwa von „Student(inn)en“, bezeichnet zwar das gleiche, wird hier aber nicht verwendet. In manchen Sprachkonstruktionen, wie zum Beispiel der obigen, könnte die verwendete Notation: „ein(e) Studierende[r]" mit gleicher Bedeutung ersetzt werden durch: „ein(e) StudentIn", bzw „ein(e) Student(in)". In diesen Fällen bedeuten die mit runden Klammern versehenen Wortteile, dass diese im gleichen Satz entweder alle verwendet oder alle gleichzeitig weggelassen werden sollen. Letztere Schreibweise werden hier in der Regel bevorzugt. Werden runde und eckige Klammern im gleichen Satz gesetzt, so bedeutet dies, dass **entweder** die mit runden Klammern **oder** die mit eckigen Klammern versehenen Teile weggelassen werden sollen.[1] Sprachliche Verrenkungen lassen sich auf diese Weise leider dennoch nicht immer vermeiden, und man (*nein*, nicht „frau", denn hier wird ja „man" auch nicht mit doppel-„n" geschrieben, selbst wenn die historischen Wurzeln vielleicht wirklich bis dorthin reichen!) möge mir nachsehen, wenn hier einer strikten geschlechtsneutralen Schreibweise nicht in jedem einzelnen Fall der absolute Vorrang eingeräumt wurde. Die Leser(innen) mögen selbst entscheiden, welche Schreibweise sie bevorzugen.

## 1.2 Einführendes Beispiel

Das folgende Bild zeigt das Innere eines Gerätes, in das eine Kugel oben bei **L** oder **R** eingeworfen werden kann. Die Hebel an den Verzweigungsstellen werden jeweils durch die Kugel beim Passieren umgestellt. Je nach dem wo die Kugel eingeworfen wird und wie dann gerade die Stellung der Hebel ist, wird die Kugel das Gerät bei dem mit **1** oder mit **0** bezeichneten Ausgang verlassen.




Um das Verhalten des Gerätes vollständig zu beschreiben, könnten wir zum Beispiel die nachstehende Liste vervollständigen:

1. Wenn in der Situation $\boxed{\substack{\backslash\;\backslash \\ \backslash}}$ die Kugel bei **L** eingeworfen wird, dann kommt sie beim Ausgang **0** heraus und die Hebelstellung danach ist: $\boxed{\substack{/\;\backslash \\ /}}$.
2. Wenn in der Situation $\boxed{\substack{\backslash\;\backslash \\ \backslash}}$ die Kugel bei **R** eingeworfen wird, dann kommt sie beim Ausgang **0** heraus und die Hebelstellung danach ist: $\boxed{\substack{\backslash\;/ \\ \backslash}}$.
3. bis 16. (analog)

[1] Anmerkung: Diese Konvention wird im Skript nicht durchgängig eingehalten. (bf)

Wir können uns die kleinen Bildchen $\boxed{\substack{\backslash\,\backslash\\ \backslash}}$ wie durch kleine Gummistempel erzeugte „Symbole“ vorstellen, und werden in diesem Sinne eine kurze Zeit damit weiterarbeiten.

Die fertiggestellte Liste wäre am Ende sicher nicht sehr übersichtlich, und wenn wir das Verhalten des Gerätes nach mehrfach wiederholtem Einwerfen der Kugel notieren sollten, wären wir wohl etwas länger beschäftigt.

Offensichtlich hängt das Verhalten des Gerätes nur davon ab, welche Schalterstellung gerade vorliegt und wo die Kugel eingeworfen wird. Zudem wird das Ergebnis dadurch offensichtlich eindeutig festgelegt.

Eine angemessenere Form der Beschreibung ist dann die einer mathematischen Funktion $\delta$, mit der Urbildmenge:

$\left\{(x_1, x_2) \mid x_1 \in \left\{\boxed{\substack{\backslash\,\backslash\\ \backslash}}, \boxed{\substack{\backslash\,/\\ \backslash}}, \boxed{\substack{\backslash\,\backslash\\ /}}, \boxed{\substack{\backslash\,/\\ /}}, \boxed{\substack{/\,\backslash\\ \backslash}}, \boxed{\substack{/\,/\\ \backslash}}, \boxed{\substack{/\,\backslash\\ /}}, \boxed{\substack{/\,/\\ /}}\right\} \text{ und } x_2 \in \{\mathbf{L}, \mathbf{R}\}\right\}$

und der Bildmenge:

$\left\{(y_1, y_2) \mid y_1 \in \left\{\boxed{\substack{\backslash\,\backslash\\ \backslash}}, \boxed{\substack{\backslash\,/\\ \backslash}}, \boxed{\substack{\backslash\,\backslash\\ /}}, \boxed{\substack{\backslash\,/\\ /}}, \boxed{\substack{/\,\backslash\\ \backslash}}, \boxed{\substack{/\,/\\ \backslash}}, \boxed{\substack{/\,\backslash\\ /}}, \boxed{\substack{/\,/\\ /}}\right\} \text{ und } y_2 \in \{\mathbf{1}, \mathbf{0}\}\right\}.$

Dies notiert sich natürlich viel leichter, wenn die Menge der Hebelstellungen besser – im Sinne einer einfacheren Notation – dargestellt wird!

Bezeichnen wir die Hebelstellung **von links oben nach rechts unten** mit der Ziffer „0“ und diejenige **von rechts oben nach links unten** mit der Ziffer „1“, so kann die Stellung der drei Hebel als Binärzahl dargestellt werden. Wir wählen die Binärzahl „001“ für die Situation $\boxed{\substack{\backslash\,/\\ \backslash}}$ und „101“ für die Situation $\boxed{\substack{/\,/\\ \backslash}}$! In Dezimaldarstellung (z.B.: $3 = [011]_2$) sind dies die Ziffern $0, 1, 2, 3, 4, 5, 6, 7$. Die acht Hebelstellungen bedeuten letztlich die möglichen internen Zustände des Gerätes, weswegen wir diese als Elemente $z_i$ der Menge $Z := \{z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$ notieren wollen. Aus dem Index $i$ des Zustands $z_i$ läßt sich ebenso eindeutig die Stellung der Hebel im Inneren des Gerätes in diesem Zustand ablesen. Die soeben natürlichsprachlich beschriebene (Übergangs-)Funktion lässt sich nun, wie in Abbildung 1.1 dargestellt, mathematisch formal notieren.

Die Übergangsfunktion haben wir hier mathematisch beschrieben durch die Zeile

$$\delta : Z \times \{L, R\} \longrightarrow Z \times \{1, 0\},$$

die ausdrückt, dass jeweils ein Zustand aus der Menge $Z$ zusammen mit einem der Symbole $L$ oder $R$ überführt werden in einen neuen Zustand – wiederum aus $Z$ – und einen der Ausgänge 1 oder 0. Hier werden also Vor- und Nachbereich der Funktion festgelegt.

$Z \times \{L, R\}$ ist (mathematisch gesehen) die Menge aller Paare aus Zustand und Eingang, während $Z \times \{1, 0\}$ die Menge aller Paare aus Zustand und Ausgang darstellt. ($\times$ bildet das **cartesische Produkt** zweier Mengen.)

In den folgenden Zeilen wird die Funktion durch explizite Angabe der Zuweisungsvorschrift spezifiziert. Die entspricht der konkreten Angabe der Menge aller Paare aus Urbild und Bild der Funktion. Dabei ist zu beachten, dass Funktionen immer rechtseindeutig sind, d.h. für jedes Element aus dem Vorbereich das Bild eindeutig bestimmt ist. Eine Funktion weist Elementen aus dem Vorbereich **deterministisch** einen Funktionswert zu.

$$
\begin{array}{rcl}
\delta : Z \times \{L, R\} & \longrightarrow & Z \times \{1, 0\} \\
(z_0, L) & \mapsto & (z_6, 0) \\
(z_0, R) & \mapsto & (z_1, 0) \\
(z_1, L) & \mapsto & (z_7, 0) \\
(z_1, R) & \mapsto & (z_2, 0) \\
(z_2, L) & \mapsto & (z_4, 1) \\
(z_2, R) & \mapsto & (z_3, 0) \\
(z_3, L) & \mapsto & (z_5, 1) \\
(z_3, R) & \mapsto & (z_0, 1) \\
(z_4, L) & \mapsto & (z_0, 1) \\
(z_4, R) & \mapsto & (z_5, 0) \\
(z_5, L) & \mapsto & (z_1, 1) \\
(z_5, R) & \mapsto & (z_6, 0) \\
(z_6, L) & \mapsto & (z_2, 1) \\
(z_6, R) & \mapsto & (z_7, 0) \\
(z_7, L) & \mapsto & (z_3, 1) \\
(z_7, R) & \mapsto & (z_4, 1)
\end{array}
$$

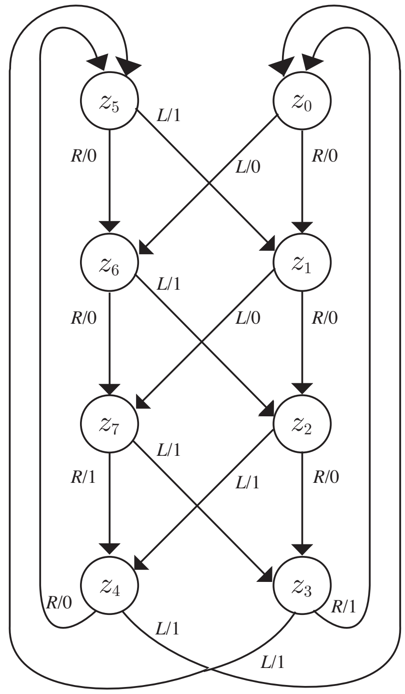



Abbildung 1.1: Übergangsfunktion (links) und Zustandsgraph (rechts)

Natürlich ist solch eine Liste von Funktionswerten nicht sehr hübsch. Etwas übersichtlicher wird es, wenn wir stattdessen einen Graph zeichnen (rechtes Bild), dessen Knoten den Zuständen zugeordnet sind und der 16 gerichtete und beschriftete Kanten für die Abbildungen besitzt. Diesen Graphen bezeichnet man gewöhnlich als **Zustandsgraph** (genauer: Zustandsübergangsdiagramm). An einer Kante steht hier die Beschriftung **L/0** (bzw. **L/1**, **R/0**, **R/1**), wenn die Kugel bei **L** (bzw. **R**) eingeworfen wird und dann bei **0** (bzw. **1**) wieder herauskommt.

Falls wir nun wissen möchten, in welchen Zustand das Gerät gelangt, wenn wir beginnend im Zustand $z_0$ die Kugel z.B. in der Folge

$$\mathbf{F} := \mathbf{L\,L\,R\,L\,R\,R\,L\,R\,L\,R\,L\,L\,R\,R\,L\,R\,L\,L\,R\,L\,R\,R\,L\,R\,R\,R\,L\,R\,R\,L\,R\,R\,R\,L\,R\,R}$$

einwerfen, so können wir dies an dem Zustandsdiagramm ablesen. Wir erhalten dabei sogar gleichzeitig die Informationen, wo die Kugel jeweils herauskommen wird. Wenn die Eingabesequenz allerdings recht lang ist, so ist dies Vorgehen sogar recht mühselig und fehleranfällig.

Bei sorgfältigem Betrachten des Zustandsdiagramms fällt auf, dass man nach viermaligem Einwerfen der Kugel bei **L** (bzw. nach viermaligem Einwerfen der Kugel bei **R**) das Gerät wieder in den gleiche Zustand versetzt bei dem man gestartet hatte. Wir notieren dies für jeden Zustand $z_i, i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ durch $z_i \overset{LLLL}{\Longrightarrow} z_i$ und $z_i \overset{RRRR}{\Longrightarrow} z_i$. Ähnlich verifizieren wir, dass für alle $i, j \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ die gleichartig notierte Beziehung $z_i \overset{RL}{\Longrightarrow} z_j$ genau dann gilt, wenn $z_i \overset{LR}{\Longrightarrow} z_j$ zutrifft. Dies bedeutet, dass wir eine lange Eingabefolge $F$ von Eingabesymbolen **L** und **R** mit dem im folgenden beschriebenen Verfahren umformen können, ohne den erreichten Zustand dabei zu verändern. Wir formulieren das Verfahren in Form von **Regeln**, die eine Folge von Aktionen (hier: Einwürfen der Kugel in die linke bzw. rechte Öffnung) in eine andere, nicht längere Folge umwandeln.

---

**Transformationsverfahren:**

Wende die folgenden Regeln solange an, wie dies möglich ist:

a) Wenn in der Folge eine der Sequenzen **L L L L** oder **R R R R** vorkommt, lösche diese.

b) Wenn in der Folge die Sequenz **R L** auftritt, ersetze diese durch die Sequenz **L R**.

---

Wenden wir diese Regeln auf die Sequenz **F** an, so ist zum Beispiel die in Abbildung 1.2 gezeigte Abfolge von Ersetzungen möglich, wobei die späteren Vertauschungsschritte bei maximaler Parallelität in einer Zeile notiert wurden. Die mit Klammern markierten Stellen deuten die Vertauschungen von Symbolen an, während unterstrichene Teilwörter entfernt werden.

$$\mathbf{LLRLRRLRLRLLRRLRLLRLRRLRRRLR\underbrace{RL}RRRLRR}$$
$$\mathbf{LLRLRRLRLRLLRRLRLLRLRRLRRRLRL\underline{RRRR}LRR}$$
$$\mathbf{LLRLRRLRLRLLRRLRLLRLR\underbrace{RL}RRRLRLLRR}$$
$$\mathbf{LLRLRRLRLRLLRRLRLLRLRL\underline{RRRR}LRLLRR}$$
$$\mathbf{LL\underbrace{RL}R\underbrace{RL}\underbrace{RL}\underbrace{RL}LR\underbrace{RL}\underbrace{RL}L\underbrace{RL}\underbrace{RL}L\underbrace{RL}LRR}$$
$$\mathbf{LLLR\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}L\underbrace{RL}\underbrace{RL}L\underbrace{RL}RR}$$
$$\mathbf{LLL\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}L\underbrace{RL}\underbrace{RL}LRRR}$$
$$\mathbf{\underline{LLLL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}L\underbrace{RL}\underbrace{RL}RRR}$$
$$\mathbf{L\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}L\underbrace{RL}\underline{RRRR}}$$
$$\mathbf{LL\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}LR}$$
$$\mathbf{LLL\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}R}$$
$$\mathbf{\underline{LLLL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}RR}$$
$$\mathbf{L\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}RRR}$$
$$\mathbf{LL\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}\underline{RRRR}}$$
$$\mathbf{LLL\underbrace{RL}\underbrace{RL}\underbrace{RL}\underbrace{RL}R}$$
$$\mathbf{\underline{LLLL}\underbrace{RL}\underbrace{RL}\underbrace{RL}RR}$$
$$\mathbf{L\underbrace{RL}\underbrace{RL}RRR}$$
$$\mathbf{LL\underbrace{RL}\underline{RRRR}}$$
$$\mathbf{LLLR}$$

Abbildung 1.2: Anwendung des Transformationsverfahrens auf die Folge **F**

Am Ende sind die erhaltenen Sequenzen stets so kurz, dass der mit einer beliebig langen Folge **F** erreichbare Zustand nun sehr leicht herauszufinden ist! Tatsächlich bestehen die so erhaltenen Folgen aus höchstens sechs Symbolen. Diese kurzen Sequenzen können mit obigen Regeln **alle** auf eine der folgenden Formen gebracht werden: $\lambda$, **L**, **R**, **L L**, **L R**, **R R**, **L L L**, **L L R**, **L R R**, **R R R**, **L L L R**, **L L R R**, **L R R R**, **L L L R R**, **L L R R R**, **L L L R R R**. Hierbei bezeichnet das griechische kleine Lambda „$\lambda$“ die Tatsache, dass nach der Umformungsprozedur keines der Symbole **L** oder **R** mehr vorhanden ist, man also in den Ausgangszustand zurückkehrt.

Offensichtlich braucht man nur die Anzahlen der vorkommenden **L**'s und **R**'s modulo 4 zu zählen, um die entsprechende kurze Folge als eine **Normalform** zu erhalten! Die vorige Folge **F** enthält 15-mal das Auftreten eines **L**'s und 21 Vorkommnisse des Symbols **R**. Modulo 4 sind das 3·**L** und 1·**R**, was vom Effekt her der Folge **L L L R** entspricht, die wir durch die Ersetzungsregeln auch erhalten hatten. Wie man nun leicht nachprüft, wird im Zustandsdiagramm mit dieser Folge **F** von jedem Zustand jeweils der benachbarte Zustand in der gleichen Höhe gegenüber eingenommen.

An diesem Beispiel zeigen sich bereits wichtige inhaltliche Bestandteile und Erkenntnisse (nicht nur) für diese Vorlesung:

- Die Notwendigkeit von **Abstraktionen** und die Beschränkung auf das Wesentliche. Es ist jedoch Vorsicht geboten, damit nicht wichtige Informationen verlorengehen!
- Die Nützlichkeit **mathematischer Notationen**, wie z.B. Funktionen und Relationen! Diese werden durchgängig in der Informatik verwendet.
- Die deutlichere Übersichtlichkeit bei Verwendung von **gerichteten und beschrifteten Graphen**. Diese werden häufig bei der Beschreibung von Automaten und anderen mathematischen Objekten verwendet. Sie dienen vorrangig der Visualisierung von (häufig) komplexen Zusammenhängen, haben jedoch immer auch eine mathematisch formale Darstellung!
- Die Möglichkeit **algorithmischer Verfahren**, zum Beispiel solche, die Transformationen von Zeichenketten benutzen. Solche Transformationen werden später im Zusammenhang mit Algorithmen und formalen Grammatiken bzw. Reduktionssystemen wie auch anderen Kalkülen untersucht.

# 2 Notationen und Grundbegriffe

Betrachten Sie folgende Aussage:

„Wenn $G = \sqrt[2]{ab}$ das geometrische Mittel (*geometric mean*) und $A = \frac{a+b}{2}$ das arithmetische Mittel (*arithmetic mean*) der positiven Zahlen $a$ und $b$ bezeichnen, so gilt stets $G \leq A$. Die Gleichheit $G = A$ gilt hier nur, wenn $a = b$ ist.“

Die hier verwendeten Gleichheitszeichen haben unterschiedliche Bedeutung: Die des ersten Satzes bedeuten die Definition der neuen Objekte $G$ und $A$ auf der Basis der schon bekannten Größen $a$ und $b$, während die im zweiten Satz Beziehungen (Relationen) bezeichnen. Wir werden diese verschiedenen Gleichheitsbegriffe auch in der Notation von einander unterscheiden. Das relationale Gleichheitszeichen „=“ wird wie üblich verwendet (Auf dem europäischen Kontinent ist es erst seit dem achtzehnten Jahrhundert in Gebrauch, obwohl das Symbol „=“ von Robert Recorde schon 1577 eingeführt und empfohlen wurde! Vergl. [Gries&Schneider], Seite 16.) und unterschieden von demjenigen, das wir benutzen um Neues aus schon Bekanntem zu definieren. Wir werden in diesem Skript für: „sei definitionsgemäß gleich“ das mit einem Doppelpunkt beginnende Gleichheitszeichen „:=“ benutzen, welches in PASCAL- und ALGOL-ähnlichen Programmiersprachen, sowie häufig bei Beschreibungen von algorithmischen Verfahren als Zuweisungssymbol verwendet wird.

Weiterhin werden neue Begriffe bei deren Definition ***fett*** gedruckt und es werden die englischen Bezeichnungen in *kursiv* (wie oben geschehen) hinter neuen Begriffen notiert. Beweise werden mit dem kleinen Quadrat „ □ “ abgeschlossen, Definitionen, Theoreme (Sätze), Lemmata (Hilfssätze) sowie Korollare (Folgerungen aus Theoremen oder Lemmata) werden dagegen ohne eigenes „Endezeichen“ geschrieben, jedoch typographisch vom restlichen Text abgesetzt.

| |
|---|
| Wichtige, dem Verständnis dienende Anmerkungen werden mit einem Rahmen versehen, der jedoch nicht die Wichtigkeit von Definitionen und insbesondere Theoremen schmälern soll! |

Wir beginnen das Skript mit einer Reihe von Definitionen und Notationen, werden aber – wo immer es geht – weitere mathematische Begriffe erst dann definieren, wenn diese aus Sicht der Anwendung und Benutzung in diesem Skript nötig werden. Dieses Kapitel ist vorwiegend als Referenz zu verstehen, in dem nachgeschlagen werden kann, welche Schreibweisen und Definitionen für dieses Skript verwendet werden, da in der Fülle der weiterführenden Literatur häufig unterschiedliche Notationen verwendet werden.

Als Begleitlektüre seien insbesondere [Hopcroft,Motwani&Ullman], [Hopcroft&Ullman] und [Kinber&Smith] empfohlen.

## 2.1 Notation von Mengen

**Mengen** Wir werden oft Mengen und darauf erklärte Relationen verwenden, und wollen hier die von uns verwendete Notationen für diese intuitiv bekannten Begriffe erklären. Cantor (Georg Cantor, 1845-1918) folgend, ist eine **Menge** (*set*) $M$ eine Zusammenfassung bestimmter, wohlunterschiedener Objekte unserer Anschauung oder unseres Denkens (welche die Elemente von $M$ genannt werden) zu einem Ganzen.

Mengen können wir auf unterschiedlichste Weise bilden: Wir können sie **extensional** (dem Umfang, der Ausdehnung nach) durch direkte Aufzählung aller Elemente zwischen den Mengenklammern „{“ und „}“ notieren, z.B. als $A := \{1, 3, 5, 7\}$ für die ersten vier ungeraden natürlichen Zahlen. Bei nicht zu umfangreichen, endlichen Mengen (*finite* [sprich: feineit] *sets*) ist das immer möglich. Unendliche Mengen (*infinite* [sprich: infinit] *sets*) könnte man etwas informal, wie im folgenden Beispiel der geraden, nicht negativen ganzen Zahlen, **aufzählend** in der Form $B := \{0, 2, 4, \cdots\}$ notieren. Diese laxe Schreibweise hat natürlich ihre Nachteile, denn es ist durchaus nicht immer klar, wie eine begonnene Aufzählung (hier z.B. 0, 2, 4,...) fortgesetzt werden soll! Sie alle kennen diese sogenannten Intelligenztests, wie sie auch bei manchen Einstellungsprüfungen vorgenommen werden.

Wie wird zum Beispiel die Folge $\langle 8, 3, 1, 5, 9, 0, 6, \ldots \rangle$ fortgesetzt?

Sie wußten, dass es sich hierbei nur um die *alphabetische Reihenfolge* der Ziffernnamen handeln konnte und nun also noch $7, 4$ und $2$ folgen werden? Herzlichen Glückwunsch!

**Folgen** von Zahlen oder anderen Objekten wollen wir hier im folgenden immer in Spitze Klammern gesetzt notieren. So bezeichnet $\langle 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots \rangle$ die Folge der bekannten Fibonacci-Zahlen $f_n, n \geq 1$ mit der rekursiven Definition durch die Rekurrenzgleichung:

$$f_n := \begin{cases} 0 & \text{falls } n \leq 0 \\ 1 & \text{falls } n = 1 \\ f_{n-1} + f_{n-2} & \text{falls } n > 1. \end{cases}$$

Diese Rekurrenzgleichung induziert natürlich sogleich ein Verfahren zum Erzeugen der einzelnen Glieder dieser unendlichen Zahlenfolge und stellt somit eine der gewünschten endlichen Repräsentationen dieser unendlichen Menge (Folge) von Objekten (der Fibonacci-Zahlen) dar. Rekursiven und induktiven Definitionen dieser und anderer Art werden Sie in der Informatik noch häufiger begegnen. Es gibt jedoch oft andere, in mancher Hinsicht günstigere Darstellungen in geschlossener Form. Für die Fibonacci-Zahlen gilt z.B. für jedes $n \in I\!N$:

$$f_n = \frac{1}{\sqrt{5}} \left( \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right).$$

Beweise für solche Charakterisierungen werden in der Diskreten Mathematik geführt. Die Bücher von Martin Aigner, ([Aigner]), und Norman L. Biggs, [Biggs], sind gute und nicht zu einfache Quellen, letzteres in englischer Sprache. In einigen, meist einfachen, Fällen werden wir solche und ähnliche Beweise in diesem Skript jedoch auch direkt aufnehmen.

**Logische Notationen** Auch wenn die Zeichen der Logik schon in der Aussagen- und Prädikatenlogik (Vorlesung F1 – Logik) detailliert behandelt wurden, wollen wir hier die sogenannten logischen Junktoren wiederholen. Wir benutzen diese jedoch meist nur bei den Definitionen von Mengen:

| | |
|---|---|
| $\neg$ | für die Negation („nicht“, (*not*) ) |
| $\wedge$ | für die Konjunktion („und“ (*and*) ) |
| $\vee$ | für die Disjunktion („oder“ (*or*) ) |
| $\rightarrow$ | für die Implikation („daraus folgt“, „impliziert“ (*implies*) ) |
| $\leftrightarrow$ | für die Biimplikation „genau dann, wenn“ (*if and only if,* Abk.: *iff*) ) |
| $\forall$ | für den Allquantor („für alle“ (*for all*) ) |
| $\exists$ | für den Existenzquantor („es gibt“ (*exists*) ) |

Die Bedeutung und die hier verwendete Notation der Quantoren sollen der Vollständigkeit halber an dieser Stelle noch einmal präzisiert werden:

- Die Formel $\forall x \in M : p(x)$ ist genau dann falsch, wenn es ein Element $x$ in $M$ gibt, für welches das Prädikat $p(x)$ nicht erfüllt ist.
- Die Formel $\exists x \in M : p(x)$ ist genau dann wahr, wenn es ein Element $x$ in $M$ gibt, für welches das Prädikat $p(x)$ erfüllt ist.

Dadurch ist der Ausdruck $\forall x \in \emptyset : p(x)$ ausnahmslos für jedes Prädikat $p$ erfüllt!

Oft werden wir Mengen aber **intensional** (der Absicht, der Zielrichtung nach) durch Angabe einer charakteristischen Eigenschaft unter Bezugnahme auf schon bekannte Mengen definieren:

Wenn $p$ eine Eigenschaft, d.h. ein Prädikat, ist, welches auf die Elemente der zu definierenden Menge $M$ zutreffen soll, schreiben wir $M := \{x \mid p(x)\}$. Um auszudrücken, dass $x$ ein Element der Menge $M$ ist, schreiben wir $x \in M$, andernfalls $x \notin M$. Unter Rückgriff auf die üblichen Notationen (siehe weiter unten) können wir nun die Mengen $A$ und $B$ von eben auch auf folgende Weise definieren:

$A := \{n \in I\!N \mid \exists m \in I\!N : 1 \leq m \leq 4 \ \wedge \ n = 2m-1\}$ und $B := \{m \mid \exists n \in I\!N : m = 2n\}$.

Die **leere Menge**, d.h., die eindeutig bestimmte Menge die kein Element enthält, könnte so definiert werden: $\emptyset := \{m \mid m \neq m\}$ oder auch als die Menge aller Drei-Pfennig Münzen nach 1948.

**Eigenschaften und Prädikate** Von Kenneth E. Iverson, (1962), dem Erfinder der Programmiersprache **APL**[1] stammt die folgende sehr nützliche Notation, mit der Wahrheitswerte von Prädikaten auf die Zahlenwerte 0 und 1 abgebildet werden:

**2.1 Definition**
*Ist $p$ ein Prädikat, so ist $[p]$ (also das $p$ in eckigen Klammern), folgendermaßen definiert:*

$$[p] := \begin{cases} 1 & \text{falls } p \text{ wahr ist} \\ 0 & \text{falls } p \text{ nicht wahr ist.} \end{cases}$$

[1] **APL** steht für **A** **P**rogramming **L**anguage

**2.2 Beispiel**

Es sind „$3 \cdot 2 = 6$“ und „$\pi = 3,14$“ zwei Prädikate für die gemäß Definition 2.1 gilt:

$$1 = [3 \cdot 2 = 6] \quad \text{und} \quad 0 = [\pi = 3,14]$$

Wenn eine Eigenschaft durch mehrere gleichzeitig geltende Prädikate $p_1, p_2, p_3, \ldots$ definiert werden soll, so vereinfachen wir bisweilen die exaktere Schreibweise und schreiben z.B. statt $\{x \mid x \in I\!N \;\wedge\; p_1(x) \;\wedge\; p_2(x) \;\wedge\; p_3(x)\}$ oft nur $\{x \in I\!N \mid p_1(x), p_2(x), p_3(x)\}$. In der Regel werden die Leser(innen) in der Literatur allerdings aussagen- oder prädikatenlogische Formeln vorfinden.

Falls für jedes $x \in M_1$ stets auch $x \in M_2$ gilt, so ist die Menge $M_1$ in der Menge $M_2$ enthalten, und wir notieren das als $M_1 \subseteq M_2$ (oder $M_2 \supseteq M_1$) wobei eben auch $M_1 = M_2$ gelten darf. In manchen Werken wird dies auch nur als $\subset$ geschrieben, aber diese Notation benutzen wir hier nicht, da diese zu leicht mit dem echten Enthaltensein verwechselt werden kann. Um auszudrücken, dass zwar $M_1 \subseteq M_2$ aber $M_1 \neq M_2$ gilt, schreiben wir $M_1 \subsetneq M_2$. Die Mengen $M_1$ und $M_2$ sind gleich, notiert als $M_1 = M_2$, wenn sowohl $M_1 \subseteq M_2$ als auch $M_2 \subseteq M_1$ gilt. Offensichtlich kommt es bei diesen Definitionen genausowenig auf die Reihenfolge, wie auf die Anzahl der Elemente der Mengen bei deren Aufschreibung an. Es gilt $\{1,3,5,7\} = \{3,1,5,5,3,7,7,1,7\}$. Die Anzahl der Elemente in einer Menge $M$ wird als **Kardinalität** (auch: Mächtigkeit) dieser Menge bezeichnet und mit $|M|$ notiert. Eine Menge ist **unendlich**, wenn sie die gleiche Kardinalität wie eine ihrer echten Teilmengen besitzt. Ist $M$ keine endliche Menge, so schreiben wir $|M| = \infty$.

**2.3 Definition**

*Sei $M \subseteq C$, dann ist die **charakteristische Funktion** von $M$ die auf ganz $C$ definierte Funktion $\chi_M : C \longrightarrow \{0,1\}$ mit $\chi_M(x) := [x \in M]$.*

Explizit notiert ergibt sich damit die übliche Schreibweise der charakteristischen Funktion:

$$\chi_M(x) := \begin{cases} 1 & \text{falls } x \in M \\ 0 & \text{sonst} \end{cases}$$

Zu jeder Menge $M$ wird durch die *charakteristische Funktion* die Zugehörigkeit von Elementen zu dieser Menge durch die Zahlen 1 und 0 ausgedrückt.

**2.4 Definition**

*Die Menge aller Teilmengen einer Menge $M$ nennt man **Potenzmenge** und wir notieren sie als $2^M$.*

**2.5 Beispiel**

Die Potenzmenge von $M := \{1,3,5,7\}$ ist:

$$\begin{aligned} 2^M \quad := \Big\{ \; & \emptyset, \{1\}, \{3\}, \{5\}, \{7\}, \\ & \{1,3\}, \{1,5\}, \{1,7\}, \{3,5\}, \{3,7\}, \{5,7\}, \\ & \{1,3,5\}, \{1,3,7\}, \{1,5,7\}, \{3,5,7\}, \{1,3,5,7\} \; \Big\} \end{aligned}$$

Jede Menge wird eindeutig durch ihre charakteristische Funktion bestimmt. Für jede Teilmenge $A$ einer Menge $M$ gilt somit $A = \{x \mid \chi_A(x) = 1\}$ und es gibt genau so viele Teilmengen von $M$ wie es Abbildungen von $M$ auf die Menge $\{0, 1\}$ gibt, da ja jede solche Abbildung eine charakteristische Funktion darstellt. Das sind aber genau $2^{|M|}$ viele. Wenn wir später Mengen von Mengen betrachten, bezeichnen wir diese meistens als **Klassen** von Mengen oder auch als **Familien** von Mengen, wenn die Klassen besondere Eigenschaften erfüllen.

Das **Komplement** einer Menge $A$ muss stets zu einer, diese enthaltenden, Obermenge $C$ gebildet werden und wir schreiben dann, $C \setminus A := \{c \mid c \in C \wedge c \notin A\}$, womit die allgemeine **Mengendifferenz** bezeichnet wird. Wenn $C$ als bekannt vorausgesetzt werden kann, schreiben wir auch $\overline{A}$ anstelle von $C \setminus A$.

$A \cup B := \{c \mid c \in A \vee c \in B\}$ bezeichnet die **Vereinigung** der Mengen $A$ und $B$, während $A \uplus B$ deren **disjunkte Vereinigung** bezeichnet, d.h. $A$ und $B$ waren schon disjunkt oder werden, durch Umbenennung der Elemente einer Menge, dazu gemacht. Der **Durchschnitt** der Mengen $A$ und $B$ wird mit $A \cap B := \{c \mid c \in A \wedge c \in B\}$ bezeichnet. Ist $M = \biguplus_{i \in I} A_i$ die disjunkte Vereinigung von Mengen $A_i$ mit $i$ aus einer nicht notwendig endlichen Indexmenge $I$, so nennt man $(A_i)_{i \in I}$ eine **Partition** von M.

Weitere Mengen, die uns häufig begegnen, sind die folgenden:

**2.6 Definition**

*Entsprechend DIN 5473 ist $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$ die Menge der **natürlichen Zahlen** (non-negative integers), also inklusive der 0, anderswo bisweilen als $\mathbb{N}_0$ notiert. Die Menge der **ganzen Zahlen** (integers) wird mit $\mathbb{Z}$ notiert. Die Menge $\mathbb{Z}^+ = \{x \in \mathbb{Z} \mid x > 0\}$ enthält genau die positiven ganzen Zahlen (ohne die 0).*

*Mit $\mathbb{Q} := \left\{ \frac{p}{q} \;\middle|\; p \in \mathbb{Z} \wedge q \in \mathbb{Z}^+ \right\}$ werden die **rationalen**, mit $\mathbb{R}$ die **reellen Zahlen** und mit $\mathbb{C}$ die **komplexen Zahlen** In den Fällen der Zahlenbereiche $M \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ werden in Analogie zu $\mathbb{Z}^+$ durch $M^+$ und $M^-$ folgende Einschränkungen bezeichnen: $M^+ := \{x \in M \mid x > 0\}$ sowie $M^- := \{x \in M \mid x < 0\}$.*

**Funktionen zur Manipulation von Zahlen** Ebenfalls von Kenneth E. Iverson stammt die Schreibweise mit den „abgefeilten“ eckigen Klammern für „die größte (bzw. kleinste) ganze Zahl kleiner (bzw. größer) als $r$“, die z.B. in den Programmiersprachen JAVA und C++, wie auch in Definition 2.7 mit den üblichen englischen Ausdrücken *floor* und *ceiling* bezeichnet werden:

**2.7 Definition**

*Für $r \in \mathbb{R}$ sei $\lfloor r \rfloor$ die größte ganze Zahl $z \in \mathbb{Z}$ mit $z \leq r$ (floor) und $\lceil r \rceil$ die kleinste ganze Zahl $z \in \mathbb{Z}$ mit $z \geq r$ (ceiling).*

Wir stellen im Folgenden einige nützliche und einfache Anwendungshilfen zusammen.

**2.8 Beispiel**

Folgende Beziehungen gelten gemäß Definition 2.7:

$$\begin{aligned}\lfloor 3.14\rfloor &= \phantom{-}3 &&= \lfloor 3.75\rfloor\\ \lfloor -3.14\rfloor &= -4 &&= \lfloor -3.75\rfloor\\ \lceil 3.14\rceil &= \phantom{-}4 &&= \lceil 3.75\rceil\\ \lceil -3.14\rceil &= -3 &&= \lceil -3.75\rceil\end{aligned}$$

Wir betrachten einige **grundlegende Eigenschaften** der *floor*- und *ceiling*-Funktionen:

$$\lfloor x\rfloor = x \quad \text{genau dann, wenn} \quad x\in\mathbb{Z} \quad \text{genau dann, wenn} \quad \lceil x\rceil = x.$$

$$x-1 < \lfloor x\rfloor \le x \le \lceil x\rceil < x+1 \qquad \text{für jedes } x$$

$$\lfloor -x\rfloor = -\lceil x\rceil, \lceil -x\rceil = -\lfloor x\rfloor \qquad \text{für jedes } x$$

$$\lceil x\rceil - \lfloor x\rfloor = [x\notin\mathbb{Z}] = \begin{cases}1 & \text{wenn } x\notin\mathbb{Z}.\\ 0 & \text{wenn } x\in\mathbb{Z}.\end{cases}$$

Außerdem können folgende **nützliche Äquivalenzen** gezeigt werden: $\forall n\in\mathbb{N}\ \forall x\in\mathbb{R}:$

$$\begin{aligned}\lfloor x\rfloor = n &\leftrightarrow n\le x<n+1\\ \lfloor x\rfloor = n &\leftrightarrow x-1<n\le x\\ \lceil x\rceil = n &\leftrightarrow x\le n<x+1\\ \lceil x\rceil = n &\leftrightarrow n-1<x\le n\\ x<n &\leftrightarrow \lfloor x\rfloor<n\\ n<x &\leftrightarrow n<\lceil x\rceil\\ x\le n &\leftrightarrow \lceil x\rceil\le n\\ n\le x &\leftrightarrow n\le\lfloor x\rfloor\end{aligned}$$

Weiterhin gilt:

$$\forall x\in\mathbb{R}, n\in\mathbb{Z}: \quad \lfloor x+n\rfloor = \lfloor x\rfloor + n$$

und

$$\forall x\in\mathbb{R}, n\in\mathbb{Z}: \quad \lceil x+n\rceil = \lceil x\rceil + n$$

**2.9 Beispiel**

Zur Übung wollen wir die folgenden Ausdrücke vereinfachen:

1. $\lceil\lfloor x\rfloor\rceil$
2. $\lceil\frac{n}{2}\rceil - \lfloor\frac{n}{2}\rfloor$

zu 1.: Da $\lfloor x\rfloor\in\mathbb{Z}$ ist, folgt natürlich sofort $\lceil\lfloor x\rfloor\rceil = \lfloor x\rfloor$.

zu 2.: Nach der letzten Gleichung der grundlegenden Eigenschaften ist für alle Zahlen $n\in\mathbb{R}$

$$\left\lceil\frac{n}{2}\right\rceil - \left\lfloor\frac{n}{2}\right\rfloor = [\text{n ist keine gerade Zahl aus } \mathbb{Z}] = \begin{cases}0 & \text{falls } n \text{ eine gerade Zahl aus } \mathbb{Z} \text{ ist,}\\ 1 & \text{sonst.}\end{cases}$$

Mehr zu diesen nützlichen Notationen und vielen weiteren Ergebnissen aus der diskreten Mathematik finden die interessierten Leser(innen) im hervorragenden Buch von R.L. Graham, D.E. Knuth und O. Patashnik: „Concrete Mathematics“, [Graham et al.].

## 2.2 Relationen

Die folgenden Begriffe zu Relationen und deren Klassifizierung nach Eigenschaften sind ebenfalls Stoff der Mathematikveranstaltung. Damit wir bei Bedarf diese Notationen benutzen können, finden Sie diese jedoch auch an dieser Stelle noch einmal wieder. Im allgemeinen wollen wir die Notationen möglichst spät und erst dann einführen, wenn die mit ihnen beschriebenen Begriffe auch benötigt werden! Das gelingt verständlicher Weise nicht durchgehend, denn es ist oft besser, ähnliche und abgeleitete Begriffe zusammenhängend zu erklären. Relationen werden wir benötigen, um Graphen und Tabellen zu notieren, einige der hier bereits eingeführten Eigenschaften werden wir aber erst später verwenden.

**2.10 Definition**

*Seien $A_1, A_2, \ldots, A_n$ Mengen und $x_1 \in A_1, \ldots, x_n \in A_n$, dann heißt $(x_1, \ldots, x_n)$ ein* ***(geordnetes) $n$-Tupel*** *von Elementen über $A_1, \ldots, A_n$. Die Menge aller geordneten $n$-Tupel $(x_1, \ldots, x_n)$ mit $x_1 \in A_1, \ldots, x_n \in A_n$ heißt* ***cartesisches Produkt*** *der Mengen $A_1$ bis $A_n$ und wird geschrieben als $A_1 \times A_2 \ldots \times A_n$.*

*Es ist also $A_1 \times A_2 \times \ldots \times A_n := \{(x_1, \ldots, x_n) \mid x_1 \in A_1, \ldots, x_n \in A_n\}$.*

| |
|---|
| Spricht man von Tupeln, so sind in der Regel immer geordnete Tupel gemeint, wenn dies nicht explizit anders angegeben wird. |

Sind die Mengen $A_i$ selbst schon cartesische Produkte der Form $A_i = B_{i,1} \times \cdots \times B_{i,k_i}$, so werden in den Elementen von $A_1 \times A_2 \times \ldots \times A_n$ die Klammern der Tupel aus $A_i$ alle weggelassen sofern dies nicht zu Mehrdeutigkeiten führen kann. Anstelle von $((a, b), (3, 2, 4), (\nabla, \Delta))$ schreiben wir also häufig einfach $(a, b, 3, 2, 4, \nabla, \Delta)$!

Eine Teilmenge $R \subseteq A_1 \times A_2 \times \ldots \times A_n$ heißt **$n$-stellige Relation** über $A_1$ bis $A_n$.

Ist $n = 2$, so spricht man häufig von einer **binären** statt von einer 2-stelligen **Relation** (genauer: heterogene dyadische Relation oder auch Korrespondenz von $A_1$ nach $A_2$).

Falls $A_i = A$ für jedes $i$ mit $1 \leq i \leq n$ ist, so sprechen wir von einer $n$-stelligen **Relation auf** $A$. Das n-fache cartesische Produkt $\underbrace{A \times A \times \ldots \times A}_{n}$ wird mit $A^n$ abgekürzt. Wenn Elemente aus $A^n$ nicht als Tupel $(x_1, x_2, \cdots, x_n)$ geschrieben und benutzt, sondern im Zusammenhang mit Matrizenrechnung in der Linearen Algebra als Vektoren gebraucht werden, so notiert man diese in der Regel als Spaltenvektoren $\vec{x} \in A^n$. Um Vektoren in Zeilendarstellung aufschreiben zu können, müssen diese transponiert werden: Es ist

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}^\top := (x_1 \; x_2 \; \cdots \; x_n)$$

und $(\vec{x}^\top)^\top = \vec{x}$. Genauso, wie die Elemente in Matrizen in der rechteckigen Anordnung ebenfalls nur durch Zwischenräume von einander getrennt werden, fehlen in dieser Darstellung die Kommata als Trennzeichen der Tupelschreibweise! Vergleichen Sie zum Beispiel auch [Möller].

Bei einer zweistelligen Relation $R \subseteq A^2$ benutzen wir anstelle von $(x, y) \in R$ oft die Infixschreibweise $xRy$ (z.B. $3 \leq 5$).

**2.11 Definition**
*Sei $R \subseteq A_1 \times A_2$ eine Relation, dann ist $A_1$ der **Vorbereich** von $R$ und $A_2$ ist der **Nachbereich**[2] von $R$. Die Menge aller Elemente $x \in A_1$, die mit einem Element $y \in A_2$ in der Relation $R$ stehen, wird als **Definitionsbereich von** $y$ (auch: Argumentbereich, oder engl. domain) bezüglich $R$ bezeichnet und als $\mathrm{dom}(y) := \{x \in A_1 \mid xRy\}$ notiert. Mit $\mathrm{dom}(R)$ wird der Definitionsbereich der gesamten Relation $R$ bezeichnet und folgendermaßen erklärt:*

$$\mathrm{dom}(R) := \bigcup_{y \in A_2} \mathrm{dom}(y) = \{x \in A_1 \mid \exists y \in A_2 : xRy\}$$

*Entsprechend ist $\mathrm{range}(x) := \{y \in A_2 \mid xRy\}$ der **Wertebereich** (engl. codomain oder range) **von** $x$ bezügich $R$. Der **Wertebereich** der gesamten Relation $R$ wird mit $\mathrm{range}(R) := \{y \in A_2 \mid \exists x \in A_1 : xRy\}$ bezeichnet.*

| Häufig wird auch $\mathrm{Def}(R)$ und $\mathrm{Bild}(R)$ anstelle von $\mathrm{dom}(R)$ und $\mathrm{range}(R)$ verwendet. |
|---|

Mit $R^{-1} := \{(y, x) \mid (x, y) \in R\}$ wird die **inverse** Relation zu $R$ bezeichnet.

Die **(Rechts-)Komposition** $R_1 \cdot R_2$ zweier Relationen $R_1$ und $R_2$ ist:

$$R_1 \cdot R_2 := \{(x, z) \mid \exists y \in \mathrm{range}(R_1) \cap \mathrm{dom}(R_2) : ((x, y) \in R_1) \wedge ((y, z) \in R_2)\}.$$

Sind $R_1, R_2$ Relationen mit $R_1 \subseteq R_2$, dann heißt $R_1$ **feiner** als $R_2$ und $R_2$ **gröber** als $R_1$.

| Das cartesische Produkt begegnet uns in der Informatik in vielen Programmiersprachen (wie z.B. PASCAL oder MODULA) als Datenstruktur **RECORD**. |
|---|

**Eigenschaften von Relationen** Die folgende Definition stellt einige wichtige Eigenschaften von Relationen zusammen. Diese Begriffe werden in vielen Bereichen der (theoretischen) Informatik verwendet und stellen somit ein Grundvokabular dar, das jedem Informatiker geläufig sein sollte.

**2.12 Definition**
*Sei $R \subseteq A \times A$ eine Relation auf $A$. $R$ heißt*

1. ***reflexiv** genau dann, wenn $(a, a) \in R$ für alle $a \in A$ gilt.*
2. ***irreflexiv** genau dann, wenn für kein $a \in A$ $(a, a) \in R$ gilt.*
3. ***symmetrisch** genau dann, wenn für $(a, b) \in R$ stets auch $(b, a) \in R$ gilt.*
4. ***asymmetrisch** genau dann, wenn aus $(a, b) \in R$ immer $(b, a) \notin R$ folgt. Eine asymmetrische Relation ist also immer irreflexiv, aber eine irreflexive Relation braucht nicht asymmetrisch zu sein!*

[2] Die Begriffe Vor- und Nachbereich werden in [Bronstein et al.] synonym mit Definitions- und Wertebereich verwendet. Wir unterscheiden die Begriffe hier, da wir z.B. auch partielle Funktionen betrachten wollen, bei denen nicht für jedes Element aus dem Vorbereich auch ein Funktionswert existieren muss.

5. ***antisymmetrisch*** *(oder identitiv), falls für alle* $a, b \in A$ *gilt:*
   $(a, b) \in R \ \wedge \ (b, a) \in R$ *impliziert stets* $a = b$.

6. ***transitiv*** *genau dann, wenn aus* $(a, b) \in R$ *und* $(b, c) \in R$ *stets* $(a, c) \in R$ *folgt. R ist also transitiv, wenn* $R \cdot R \subseteq R$ *gilt.*

7. ***linear*** *genau dann, wenn für beliebige* $a, b \in A$ *genau eine der drei Bedingungen:* $a = b$ *oder* $(a, b) \in R$ *oder* $(b, a) \in R$ *erfülllt ist.*

Aus diesen elementaren Eigenschaften von Relationen kann man für spezielle Typen von Relationen Zusammenstellungen solcher Eigenschaften betrachten. Ein spezieller Relationentyp, dem besondere Bedeutung zukommt, sind die Äquivalenzrelationen. Sie werden in Kapitel 3 benötigt und werden durch Reflexivität, Symmetrie und Transitivität charakterisiert. Weitere wichtige Relationentypen werden eingesetzt, um die Elemente von Mengen anzuordnen oder wenigstens miteinander vergleichen zu können. Hierzu gehören u.a. Größenvergleiche, wie $\leq$ und $\geq$ auf Zahlenmengen.

| Eine „nicht symmetrische“ Relation muss weder „asymmetrisch“ noch „antisymmetrisch“ sein! |
|---|

**Ordnungsrelationen** So wie die Relation „kleiner“ auf den natürlichen Zahlen $\mathbb{N}$, werden Relationen im allgemeinen als Ordnungsrelationen bezeichnet, wenn sie gewisse „ordnende“ Eigenschaften erfüllen. Die meisten dieser Notationen werden auch in den entsprechenden Mathematik-Veranstaltungen erklärt, oder sind schon aus der Schule bekannt. Leider gibt es bei den Definitionen auch in der Literatur einige Unstimmigkeiten über die Bezeichnungen, die sich zum einen mit der Zeit geändert haben, bzw. in anderen Ländern mit leichten Unterschieden verwendet werden. Wir wollen hier die Bezeichnungen so verwenden, wie sie sich auch in den meisten Lehrbüchern wiederfinden.

**2.13 Definition**
*Eine Relation* $R \subseteq A \times A$ *heißt*

1. ***Quasiordnung*** *(quasi order) oder auch Vorordnung bzw. Präordnung (preorder) genau dann, wenn sie* reflexiv *und* transitiv *ist.*[3]

   *Sei K eine Teilmenge von A, d.h.* $K \subseteq A$*. Ein Element* $u \in A$ *heißt* ***untere Schranke*** *von K genau dann, wenn* $\forall x \in K : (u, x) \in R$ *gilt. Die Menge aller unteren Schranken von K bezeichnen wir als* $\mathrm{MI}(K)$.

   *Ein Element* $o \in A$ *wird* ***obere Schranke*** *genannt, wenn* $\forall x \in K : (x, o) \in R$ *gilt. Die Menge aller oberen Schranken von K bezeichnen wir als* $\mathrm{MA}(K)$.

2. ***partielle Ordnung*** *(partial order) genau dann, wenn sie* antisymmetrisch, reflexiv *und* transitiv *ist, d.h. wenn sie eine antisymmetrische Quasiordnung ist.*[4] *Die Menge A heißt dann partiell geordnet bzgl. R und wird kurz* **poset** *(**p**artially **o**rdered **set**) genannt.*

[3] In der Literatur findet man machmal auch den Begriff der irreflexiven Quasiordnung, für die neben der Transitivität nur Irreflexivität gelten muss. In diesem Fall würde die hier definierte Quasiordnung zur Unterscheidung als reflexive Quasiordnung bezeichnet. Wir bezeichnen irreflexive Quasiordnungen als *Striktordnung.* Zum Beispiel in [Gries&Schneider] wird jedoch der Begriff *quasi order* gerade für solch transitive und irreflexive Relationen verwendet!

[4] Analog zur vorigen Fußnote zu Quasiordnungen, wird auch manchmal zwischen reflexiven und irreflexiven **Halbordnungen** unterschieden. Eine partielle Ordnung in unseren Sinne ist dann eine reflexive Halbordnung. Zum Beispiel in [Floyd&Beigel]) wird für eine partielle Ordnung jedoch die Reflexivität *nicht* gefordert und nur Transitivität und Antisymmetrie vorausgesetzt.

*$a \in A$ heißt **minimal** (least) (bzw. **maximal**(maximal)) bzgl. $R$ genau dann, wenn es **kein** $b \in A$ gibt mit $a \neq b$ und $(b,a) \in R$ (bzw. $(a,b) \in R$).*

*Ein Element $a \in A$ heißt **Minimum** (bzw. **Maximum**) wenn $\forall b \in A \setminus \{a\} : (a,b) \in R$ (bzw. $\forall b \in A \setminus \{a\} : (b,a) \in R$) gilt.*

3. ***Halbordnung** genau dann, wenn sie asymmetrisch und transitiv ist.* [5]

   *Sei $K \subseteq A$. Als **Infimum**, auch größte untere Schranke (**g**reatest **l**ower **b**ound) oder untere Grenze, bezeichnet man eine untere Schranke $u \in \mathrm{MI}(K)$, wenn $\forall x \in \mathrm{MI}(K) : (x \neq u \rightarrow (x,u) \in R)$ gilt. Ein Infimum ist also das Maximum der Menge der unteren Schranken von $K$ bzgl. $R$. Falls ein Infimum existiert, so ist es eindeutig und wird mit $\inf(K)$ (oft auch $\bigwedge K$) bezeichnet.*

   *Als **Supremum**, auch kleinste obere Schranke (**l**east **u**pper **b**ound) oder obere Grenze, bezeichnet man eine obere Schranke $o \in \mathrm{MA}(K)$, wenn $\forall x \in \mathrm{MA}(K) : (x \neq u \rightarrow (o,x) \in R)$ gilt. Ein Supremum ist also das Minimum der Menge der oberen Schranken von $K$ bzgl. $R$. Falls ein Supremum existiert, so ist es eindeutig und wird mit $\sup(K)$ (oft auch mit $\bigvee K$) bezeichnet.*

4. ***Striktordnung** (Abk. **spo** für **s**trict **p**artial **o**rder)[6], falls sie asymmetrisch und transitiv ist. Jede irreflexive und transitive Relation ist asymmetrisch, so dass eine Striktordnung auch als transitive, irreflexive Relation definiert werden könnte. Der **strikte** Teil einer partiellen Ordnung $R \subseteq A \times A$ ist $R \setminus Id_A$, wobei $Id_A := \{(a,a) \mid a \in A\}$ die **Identitätsrelation** auf $A$ bezeichnet.*

5. ***lineare** (oder **totale**) **Ordnung** genau dann, wenn sie partielle Ordnung ist und zusätzlich für alle $a,b \in A$ mit $a \neq b$ entweder $(a,b) \in R$ oder $(b,a) \in R$ gilt, d.h. $R$ ist konnex. Man sagt, dass die Menge $A$ total geordnet ist.*

6. ***wohl-fundiert** (well founded), wenn jede nicht leere Teilmenge von $A$ ein minimales Element besitzt.*

7. ***Wohlordnung** (well ordering), wenn sie eine wohl-fundierte lineare Ordnung ist.*

**2.14 Beispiel**

Die Relation $\leq$ auf $\mathbb{Z}$ ist eine lineare Ordnung ohne minimale oder maximale Elemente, während auf $I\!N$ mit $\leq$ total geordnet ist, als Minimum $0$ hat, aber kein maximales Element besitzt. $\leq$ ist eine Wohlordnung auf $I\!N$, nicht jedoch auf $\mathbb{Z}$.

Die Menge $K := \{x \in \mathbb{Q} \mid x^2 < 2\}$ besitzt keine obere Schranke, d.h., $sup(K) = \emptyset$.

$(I\!N, <)$ ist kein poset, denn $<$ ist nicht reflexiv. Gleiches gilt für die Relation **verheiratet** $\subseteq (M \cup F) \times (M \cup F)$, wobei $M$ die Menge der Männer und $F$ die der Frauen abkürzt.

Die Relation $div \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$, definiert durch $(m\ div\ n) \leftrightarrow (\frac{n}{m} \in \mathbb{Z}^+ \setminus \{1\})$, d.h., $m$ ist echter Teiler von $n$, ist eine Striktordnung (aber keine totale Ordnung) und besitzt $1$ als Minimum, sofern man auch die Zahl $1$ als echten Teiler bezeichnet (siehe Abbildungen auf Seite 19).

Für jede Menge $M$ ist $(2^M, \subseteq)$ eine partiell geordnete Menge.

Für partielle Ordnungen werden oft Zeichen wie $\leq$, $\subseteq$, $\sqsubseteq$ oder $\preceq$ verwendet, während $<$, $\subset$, $\sqsubset$ oder $\prec$ für Striktordnungen benutzt werden.

[5] Eine Halbordnung ist also gerade eine asymmetrische, irreflexive Quasiordnung.

[6] auch *asymmetrische (irreflexive) Halbordnung* genannt

Um **endliche** Halbordnungen (partielle Ordnungen oder Striktordnungen) darzustellen werden häufig „Hasse-Diagramme“ als graphische Darstellungen verwendet.

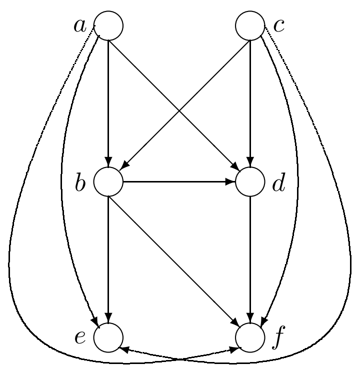
**2.15 Definition**

*Das **Hasse-Diagramm** zu einer endlichen, asymmetrischen oder antisymmetrischen, transitiven Relation $R$, ist der gerichtete Graph $G(R_0)$ einer Teilmenge $R_0 \subseteq R$ mit folgender Eigenschaft: $(a,b) \in R_0$ gilt genau dann, wenn $(a,b) \in R$ und für kein Element $c \in \mathrm{dom}(R) \cup \mathrm{range}(R)$ mit $b \neq c \neq a$ sowohl $(a,c) \in R$ als auch $(c,b) \in R$ gilt. Gerichtete Kanten in $G(R_0)$ der Art $a \longrightarrow b$ werden also genau dann gezeichnet, wenn $(a,b) \in R$ ist und „kein weiterer Knoten $c$ dazwischen liegt“.*

Hasse-Diagramme werden häufig nur zu Striktordnungen definiert, jedoch ist eine Erweiterung auf alle transitiven Relationen möglich, nur nicht ganz so einfach zu beschreiben. Für Striktordnungen $R \subseteq A \times A$ ist das Hasse-Diagramm $G(R_0)$ gegeben durch $R_0 := (R \setminus Id_A) \setminus (R \setminus Id_A)^2$. In der Regel zeichnen Mathematiker in Hasse-Diagrammen keine Pfeile, sondern ungerichtete Kanten. Die Richtung wird durch die Platzierung der Knoten gegeben: Geht nach unserer Definition eine Kante vom Knoten $a$ zum Knoten $b$, so wird $a$ oberhalb von $b$ gezeichnet.

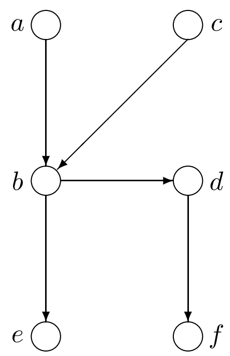
Hasse-Diagramm für die folgende Striktordnung:

$R := \{\ (a,b),(a,d),(a,e),(a,f),(b,d),(b,e),(b,f),(c,b),(c,d),(c,e),(c,f),(d,f)\ \}.$

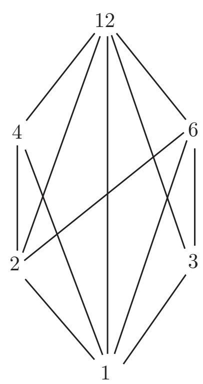



Graph der Striktordnung $R$

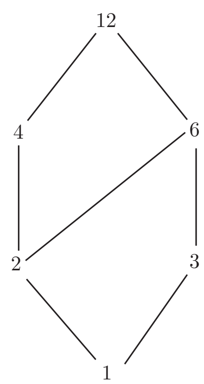
Wollten wir den vollständigen Graph einer Quasiordnung zeichnen, so müssten an alle Knoten noch Schleifen für die Reflexivität angefügt werden.




Hasse-Diagramm zu $R$




Graph der Striktordnung *div*




Hassediagramm zur Striktordnung *div*

Die Striktordnung *div* wurde als „echter-Teiler-von“ Relation auf Seite 18 definiert

**Abschlussoperatoren** Das Hasse-Diagramm ist der Graph einer in der Regel kleineren (Paar-)Menge, welche die gesamte Striktordnung durch transitiven Abschluss, bzw. eine partielle Ordnung durch reflexiven und transitiven Abschluss erzeugt.

**2.16 Definition**

*Sei $R \subseteq A \times A$ eine Relation auf der Menge $A$. Dann seien $R^+$, der* ***transitive Abschluss*** *(auch* ***transitive Hülle****), und $R^*$, der* ***reflexive, transitive Abschluss****, von $R$ wie folgt erklärt:*

$$R^+ := \bigcup_{i \geq 1} R_i \text{ mit } R_1 := R \quad \text{und} \quad R_{i+1} := R_i \cdot R. \qquad R^* := R^+ \cup Id_A.$$

Hier ist entsprechend Definition 2.13 $Id_A := \{(a, a) \mid a \in A\}$ die *Identitätsrelation* auf $A$ und offensichtlich $R_i$ die $i$-fache Komposition von $R$ mit sich selbst, was nicht identisch mit $R^i = \underbrace{R \times R \times \cdots \times R}_{i-mal}$ ist!

Betrachten wir das Geschehen bei den Relationen und deren Graphen, so bilden wir von der Menge $R_0 := \{(a, b), (c, b), (b, d), (b, e), (d, f)\} \subseteq R$, deren Graph $G(R_0)$ gerade ein Hasse-Diagramm der halbgeordneten Menge $R$ ist, den **transitiven Abschluss**, um $R$ zu erhalten.

**2.17 Beispiel**

Die Quasiordnung $R_c := \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, c), (c, b), (c, a)\}$ besitzt verschiedene erzeugende Relationen $R_e$ mit $R_e^+ = R_e^* = R_c$. Zwei davon sind z.B.: $R_e := \{(a, b), (b, c), (c, a)\}$ und $R_e := \{(a, b), (b, a), (b, c), (c, b)\}$.

Nun ist jedoch nach obiger Definition der transitive Abschluss einer Relation $R$ als unendliche Vereinigung definiert worden, und das scheint für algorithmische Verfahren kein gangbarer Weg zu sein. Muss hier jedoch wirklich eine unendliche Vereinigung gebildet werden?

Wenn die Grundmenge $A$ und auch range($R$) unendliche Mengen sind, so kann es passieren, dass tatsächlich eine unendliche Vereinigung gebildet werden muss. Wir werden sehen, dass dies nicht erforderlich ist, wenn $R$ oder sogar nur range($R$) endlich ist!

**2.18 Beispiel**

Wir betrachten $G \subseteq \mathbb{Z} \times \mathbb{Z}$ mit $G := \{(x, 2x) \mid x \in \mathbb{Z}\}$. Offensichtlich ist $(x, 2^i x) \in G_i \setminus G_{i-1}$ für jedes $i \in I\!N$. Wenn jedoch die Menge $A$ endlich ist, so kann es in $R^+$ höchstens $r := |A|^2$ viele verschiedene Elemente geben. Für jedes $i \in I\!N$ enthält die Menge $R_i$ davon eine Auswahl, und auch von diesen möglichen Auswahlen gibt es nur begrenzt viele, nämlich maximal so viele wie die Anzahl aller verschiedenen Teilmengen von $A \times A$, die genau $\sum_{j=0}^{r} \binom{r}{j} = 2^r$ ist. (Einen kurzer Beweis für diese Summe folgt aus der binomischen Formel: $\forall a, b \in I\!R : (a + b)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i$. Setzt man in dieser Formel $a = b = 1$, so folgt die zu beweisende Formel sofort.)

Also kann die unendliche Vereinigung $\bigcup_{i \geq 1} R_i$ aus nur endlich vielen verschiedenen Mengen gebildet werden!

**Schubfachprinzip** Ohne eine formale Analyse der Eigenschaften der Mengen $R_i$ aus Beispiel 2.18 zu machen beweisen wir diese Endlichkeitseigenschaft exakter. Wir stellen dazu zunächst fest, dass die Menge $R_i$das Element $(a, b)$ genau dann enthält, wenn es im Graphen $G(R)$ einen Pfad der Länge $i$ von $a$ nach $b$ gibt! Wir werden als Beweisprinzip das wichtige Schubfachprinzip (*box principle* oder auch *pigeonhole principle*) von Lejeune Dirichlet (1805-1859) in seiner einfachsten Variante verwenden.

Das **Schubfachprinzip** in seiner einfachsten Formulierung sagt, dass jedes Verteilen von mehr als $k$ Objekten in $k$ Kästen dazu führt, dass in einem Kasten mindestens zwei Objekte liegen werden. Etwas mathematischer formuliert lautet es so:

**2.19 Theorem (Schubfachprinzip, einfache Variante)**
*Sind $A$ und $B$ Mengen mit $|A| > |B|$, so gibt es keine injektive Abbildung $f : A \longrightarrow B$.*

Betrachten wir einen Vektor $\vec{x} := (x_1, x_2, \ldots, x_n)^\top \in \mathbb{R}^n$, so kann dieser auch als eine Abbildung $k$ der $n$ Komponenten von $\vec{x}$ auf die Werte $x_i \in \mathbb{R}$ angesehen werden:

$$k : \{1, \ldots, n\} \longrightarrow \mathbb{R} \quad \text{mit} \quad k(i) := \begin{cases} x_1 & \text{falls } i = 1 \\ \vdots & \\ x_n & \text{falls } i = n \end{cases}$$

Die Komponenten können wir als $n$ Schubfächer ansehen, in welche die reelle Zahl $\sum_{i=1}^{n} x_i$ auf irgendeine Weise aufgeteilt wird. Wenn wir die Größen $\text{mean}(\vec{x})$ für das arithmetische Mittel und $\max(\vec{x})$ für das Maximum der Komponenten von $\vec{x}$ benutzen, so läßt sich eine leichte Verallgemeinerung des Schubfachprinzips für reelle Zahlen formulieren.

**2.20 Definition**
*Sei $n \in \mathbb{N}$ und $\vec{x} := (x_1, x_2, \ldots x_n)^\top \in \mathbb{R}^n$ dann bezeichne*

$$\text{mean}(\vec{x}) := \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

*das* ***arithmetische Mittel*** *(mean) und*

$$\max(\vec{x}) := \max\{x_i \mid 1 \leq i \leq n\}$$

*das* ***Maximum*** *der Komponenten von $\vec{x}$.*

*Ist $M \subseteq \mathbb{R}$ eine endliche Menge, so ist*

$$\text{mean}(M) := \frac{1}{|M|} \cdot \sum_{x \in M} x$$

*und*

$$\max(M) := \max\{x \mid x \in M\}.$$

Mit diesen Abkürzungen gilt nun folgendes Theorem:

**2.21 Theorem (Schubfachprinzip, leicht verallgemeinert)**
*Für jedes $n \in \mathbb{N}$ und jeden beliebigen Vektor $\vec{x} \in \mathbb{R}^n$ gilt:*

$$\text{mean}(\vec{x}) > 1 \qquad \textit{impliziert} \qquad \max(\vec{x}) > 1.$$

Wenn in Theorem 2.21 $\vec{x} \in \mathbb{N}^n$ gewählt ist, so ist das (arithmetische) Mittel (*mean*) bzw. der durchschnittliche Wert (*average*) $\frac{1}{n} \cdot \sum_{i=1}^{n} x_i$ nur dann größer *1*, falls mehr als $n$ Objekte mit dem Gesamtwert $\sum_{i=1}^{n} x_i$ auf die $n$ Fächer verteilt werden. Also folgt die einfache Variante des Schubfachprinzips aus dieser leichten Verallgemeinerung.

In vielen Fällen ist eine weitere Verallgemeinerung des einfachen Schubfachprinzips nützlich, die wir hier nicht nur formulieren, sondern auch beweisen wollen, selbst wenn dies alles offensichtlich erscheint.

**2.22 Theorem (verallgemeinertes Schubfachprinzip)**

1. *Für jedes $n \in \mathbb{N}$ und jeden beliebigen Vektor $\vec{x} \in \mathbb{R}^n$ gilt:* $\mathrm{mean}(\vec{x}) \leq \max(\vec{x})$.

2. *Für jedes $n \in \mathbb{N}$ und jeden beliebigen Vektor $\vec{x} \in \mathbb{N}^n$ gilt:* $\lceil \mathrm{mean}(\vec{x}) \rceil \leq \max(\vec{x})$.

**Beweis:** Aus der Definition des arithmetischen Mittels und des Maximums folgt sofort $\mathrm{mean}(\vec{x}) = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \leq \frac{1}{n} \cdot \sum_{i=1}^{n} \max(\vec{x}) = \frac{1}{n} \cdot n \cdot \max(\vec{x}) = \max(\vec{x})$. Somit gilt *1*. Falls nur natürliche Zahlen verwendet werden, so ist auch $\max(\vec{x}) \in \mathbb{N}$ und *2* folgt direkt aus *1*. □

Es folgt aus dem verallgemeinerten Schubfachprinzip auch stets das einfache Schubfachprinzip, nicht jedoch umgekehrt! Nach dem verallgemeinerten Schubfachprinzip können wir $\mathrm{mean}(\vec{x}) \leq \max(\vec{x})$ voraussetzen. Wenn nun $\mathrm{mean}(\vec{x}) > 1$ gilt, ergibt sich $1 < \mathrm{mean}(\vec{x}) \leq \max(\vec{x})$ und also $\max(\vec{x}) > 1$ wie in Theorem 2.21 behauptet. Aus Theorem 2.21 folgt aber, wie gezeigt, auch das einfache Schubfachprinzip, Theorem 2.19.

In manchen Fällen ist die Anwendung des Schubfachprinzips nicht ganz einfach, weil nicht sofort klar ist, was den „Objekten“ (Menge $A$ aus Theore 2.19) und und was den „Schubladen“ (Menge $B$ aus Theore 2.19) entsprechen soll. Daher geben wir hier zwei Beispiele an.

**2.23 Beispiel**

Sei $A \subseteq \mathbb{Z}^+$ eine Menge von zehn beliebigen natürlichen Zahlen nicht größer als 106, d.h. $A \subseteq D := \{1, \ldots, 106\}$ mit $|A| = 10$. Es gibt dann immer zwei disjunkte Teilmengen $B, C \subseteq A$ mit $\sum_{x \in B} x = \sum_{x \in C} x$.

**Beweis (zu Beispiel 2.23):** Es gibt $2^{10} = 1024$ viele unterschiedliche Teilmengen, die wir mit $A_i \in 2^A, 0 \leq i \leq 1023$ bezeichnen. Die leere Menge $A_0 := \emptyset$ wird als eine der möglichen Lösungsmengen $B$ oder $C$ natürlich niemals vorkommen können, da $\sum_{x \in \emptyset} x = 0 \neq \sum_{x \in C} x$ für jede nichtleere Menge $C$ ist.

Wir definieren $f : 2^A \setminus \emptyset \longrightarrow \mathbb{N}$ durch $f(A_i) := \sum_{x \in A_i} x$, und ordnen damit jeder nichtleeren Teilmenge von $A$ ihre Elementsumme zu. Keine Summe von höchstens zehn verschiedenen Zahlen $n_j \in \{1, \ldots, 106\}$ überschreitet jemals den maximalen Wert 1015. Folglich ist $f(A_i) \leq 1015$ für jedes $i \in \{1, \ldots, 1023\}$ und somit gilt $|\mathrm{range}(f)| \leq 1015$.

Die Abbildung $f : 2^A \setminus \emptyset \longrightarrow \mathbb{N}$ kann nun wegen $|\mathrm{dom}(f)| = 1023$ nach dem einfachen Schubfachprinzip (Theorem 2.19) nicht injektiv sein und es muss daher zwei verschiedene Indizes $p, q \in \{1, \ldots, 1023\}$ mit $f(A_p) = f(A_q)$ geben. Da diese Mengen jedoch nicht disjunkt sein müssen, bilden wir $B := A_p \setminus (A_p \cap A_q) = A_p \setminus A_q$ und $C := A_q \setminus (A_p \cap A_q) = A_q \setminus A_p$. Die Mengen $B$ und $C$ sind nun disjunkt und aus $f(A_p) = f(A_q)$ folgt sogleich $f(B) = f(A_p) - f(A_p \cap A_q) = f(A_q) - f(A_p \cap A_q) = f(C)$. □

Bei dem Versuch, die Menge $D$ der zugelassenen Zahlen um die Zahl 107 zu erweitern, also $D := \{1, \ldots, 107\}$ zu erlauben, wird das Argument fehlschlagen, da dann $\text{dom}(f) = 1023 < \text{range}(f) = 1025$ wird. Solch eine Menge $A$ zu finden, für die es keine Partition der beschriebenen Art gibt, ist jedoch nicht ganz einfach, auch wenn wir wissen, dass es diese geben *könnte*! Die Tatsache, dass wir das Schubfachprinzip nicht für einen Beweis verwenden können, besagt aber nicht, dass es nicht möglicherweise eine andere Beweistechnik geben könnte, mit der ein Beweis gelingt

**Allgemein:** Wenn ein Beweisverfahren scheitert, so kann es immer noch in anderes geben, das erfolgreich angewandt werden kann – es sei denn man kann **beweisen, dass jedes Verfahren zum Scheitern verurteilt ist!** Ein solcher Beweis ist meist sehr schwierig.

Selbst wenn wir wissen, dass als Summe 1 und 2 bei disjunkten Mengen $B$ und $C$ nicht auftreten können, wären die Mengen $\text{dom}(f)$ und $\text{range}(f)$ noch gleich mächtig und das Schubfachprinzip würde immer noch nicht greifen!

**2.24 Beispiel**

Sei $\langle n_1, n_2, \ldots, n_k \rangle$ eine Folge von $k$ nicht notwendigerweise verschiedenen ganzen Zahlen. Dann gibt es immer eine Teilfolge, deren Summe ohne Rest durch $k$ teilbar ist.

**Beweis (zu Beispiel 2.24):** Wir betrachten für jedes $i \in \{1, \ldots, k\}$ die Teilsummen $s_i := \sum_{j=1}^{i} n_j$. Wenn schon eine dieser Teilsummen ohne Rest durch $k$ teilbar ist, sind wir fertig. Ist das nicht der Fall, so müssen die Reste $\textbf{mod}\, k$ alle ungleich 0 sein. Seien nun $r_i \equiv s_i \,\textbf{mod}\, k$ mit $1 \leq r_i \leq k-1$ alle echten Reste bei Division durch $k$.

Da es nur $k-1$ solche Reste gibt, aber $k$ unterschiedliche Teilsummen $s_i$ gebildet wurden, muss es zwei Teilsummen $s_p$ und $s_q$ geben, die den gleichen Rest $\textbf{mod}\, k$ haben.

O.B.d.A. (**o**hne **B**eschränkung **d**er **A**llgemeinheit) sei $p < q$. Mit $s_p \equiv s_q \,\textbf{mod}\, k$, ist $n_{p+1} + \cdots + n_q$ eine durch $k$ teilbare Zahl! Hier wurde wieder das einfache Schubfachprinzip angewendet. □

**2.25 Theorem**

*Sei $A$ eine endliche Menge und $R \subseteq A \times A$. Dann gibt es $t \in \mathbb{N}$, so dass $R^+ = \bigcup_{i=1}^{t} R_i$ ist.*

**Beweis:** Aus Definition 2.18 wissen wir, dass $R^+ := \bigcup_{i \geq 1} R_i$ ist.

Betrachten wir die Mengen $R_i$ für jedes $i \geq 1$. Da es höchstens $2^{|A|^2} - 1$ viele verschiedene nichtleere Mengen $R_i$ geben kann, muss in der Folge $R_1, R_2, R_3, \ldots R_{2^{|A|^2}}$ eine Menge $R_j,\ j \geq 1$, zweimal vorkommen. (Anwendung des einfachen Schubfachprinzips).

Trete z.B. die Menge $R_j = R_l$ in dieser Aufzählung nach $k$ Schritten das erste Mal wieder als Menge $R_l := R_{j+k}$ auf. Mit $R_j = R_{j+k}$ gilt also auch $\forall m \in \mathbb{N} : R_{j+m} = R_{j+k+m}$. Also folgt

$$R^+ = \bigcup_{i \geq 1} R_i = \bigcup_{i=1}^{t} R_i \text{ für } t := j + k < 2^{|A|^2}.$$

□

Die unendliche Vereinigung $R^+ = \bigcup_{i \geq 1} R_i$ kann also als endliche Vereinigung von effektiv konstruierbaren Mengen $R_i$ ebenfalls in endlich vielen Schritten konstruiert werden!

Für Beweise sind bisweilen statische Beschreibungen, wie jene, die sich aus dem folgenden Theorem ergibt, leichter zu handhaben. Für algorithmische Verfahren liefern diese Beschreibungen und Beweise jedoch leider nur sehr selten Hinweise zu deren Umsetzung.

**2.26 Theorem**

*Sei $R$ eine Relation auf $A$, $R^*$ wie in Definition 2.16 erklärt, und außerdem sei $Q_A := \{Q \mid Q$ ist Quasiordnung auf $A\}$ die Klasse aller reflexiven und transitiven Relationen auf $A$. Dann gilt:*

$$R^* = \widehat{R} := \bigcap_{\substack{R \subseteq Q \\ Q \in Q_A}} Q.$$

Das Theorem besagt in anderen Worten: Der reflexive, transitive Abschluss $R^*$ einer Relation $R$ ist gerade der Durchschnitt aller Quasiordnungen, die $R$ enthalten. $R^*$ ist also die kleinste Quasiordnung, die $R$ enthält.

**Beweis (zu Theorem 2.26):** Zunächst ist $\widehat{R}$ als Durchschnitt von reflexiven transitiven Relationen wieder reflexiv und transitiv. Weiter ist $R \subseteq \widehat{R} \subseteq R^*$ denn $R^*$ ist reflexiv und transitiv nach Definition 2.16 und daher auch Element der Klasse $Q_A$. Da $\widehat{R}$ Durchschnitt von anderen Mengen mit $R^*$ ist, kann $\widehat{R}$ nur kleiner sein. Zu zeigen ist nur noch $R^* \subseteq \widehat{R}$, was sofort aus den Inklusionen $R^* \subseteq Q$ für **jedes** einzelne $Q$ des Durchschnitts folgt. Dies zeigen wir durch Induktion über den Index $i$ der Mengen $R_i$ in der Definition von $R^* = R^+ \cup Id_A$. Die Identität $Id_A = R_0$ ist selbstverständlich in jeder reflexiven Relation $Q \in Q_A$ enthalten. Als Induktionsannahme sei $R_j \subseteq Q$ für jedes $Q \in Q_A$. Sei $(a, c) \in R_{j+1} = R_j \cdot R$, dann $\exists b \in A: \ (a, b) \in R_j \ \wedge \ (b, c) \in R$. Nach Definition des Durchschnittes zur Bestimmung von $\widehat{R}$ ist $R \subseteq Q$ und nach Induktionsannahme gilt auch $R_j \subseteq Q$. Da $Q$ transitiv ist folgt sofort $R_{j+1} \subseteq Q$. Mithin gilt $R_i \subseteq Q$ für jedes $i$ und folglich ist ihre unendliche Vereinigung $R^*$ ebenfalls Teilmenge von jedem $Q \in Q_A$ des Durchschnitts. Daher ergibt sich wie behauptet $R^* \subseteq \widehat{R}$ und insgesamt die Gleichheit. □

$R^*$ ist zwar stets eine Quasiordnung, aber nicht immer eine partielle Ordnung. Als ein einfaches Gegenbeispiel wählen wir $R := R_c$ des Beispiels auf Seite 20.

## 2.3 Funktionen

Relationen sind als beliebige Verknüpfungen zweier Mengen definiert worden, wobei nur eine Richtung der Kanten vorgegeben ist. Spezielle Relationen sind uns als **Abbildungen** oder **Funktionen** bekannt. In Abbildung 2.1 sei zunächst eine grafische Übersicht über die wichtigsten Begriffe im Zusammenhang mit Funktionnen und Relationen gegeben. Im Folgenden werden diese Eigenschaften formal definiert.

**2.27 Definition**

*Eine Relation $R \subseteq A \times B$ heißt*

1. ***rechtseindeutig** oder **(partielle) Funktion** genau dann, wenn aus $(a, b) \in R \ \wedge \ (a, c) \in R$ stets $b = c$ folgt. Das heißt, dass zu jedem $a \in A$ höchstens ein $b \in B$ existiert mit $(a, b) \in R$. Dieses Element werden wir dann mit $R(a)$ bezeichnen. Wir notieren Funktionen meist mit kleinen Buchstaben (z.B. $g, f$) und wie üblich durch Angabe von Argument- und Wertebereich in der Form $f : A \longrightarrow B$, wenn $f$ total ist und durch $f : A \underset{p}{\longrightarrow} B$, wenn $f(a)$ möglicherweise nicht für jedes $a \in A$, also nur partiell, definiert ist. Wenn $A$ ein cartesisches Produkt von $n$ Mengen ist, so heißt $n$ die **Stelligkeit** der Funktion.*

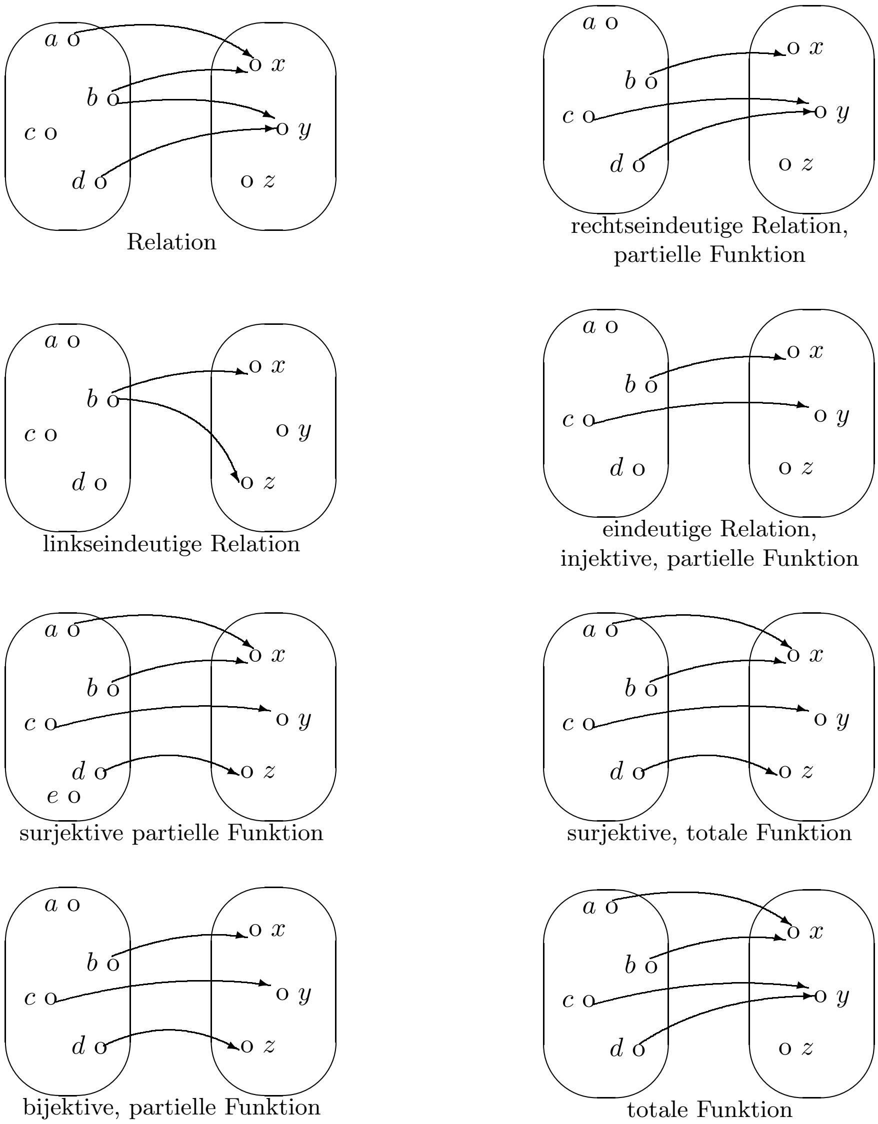



Abbildung 2.1: Eigenschaften von Funktionen und Relationen

2. ***linkseindeutig*** *genau dann, wenn aus* $(a, b) \in R \wedge (c, b) \in R$ *stets* $a = c$ *folgt.*

3. ***linkstotal*** *oder* ***total*** *genau dann, wenn* $\mathrm{dom}(R) = A$ *ist.*

4. ***Abbildung*** *genau dann, wenn* $R$ *eine totale Funktion ist, d.h.* $|\mathrm{range}(a)| = 1$ *für jedes* $a \in A$ *ist.*

Hier sollen also Abbildungen, im Gegensatz zu Funktionen, **nicht** partiell sein!

Wenn $f : A \underset{p}{\longrightarrow} B$ eine Funktion mit $A = A_1 \times A_2 \times \cdots \times A_n$ ist, so schreiben wir anstelle von $f((x_1, x_2, \ldots, x_n))$ kürzer $f(x_1, x_2, \ldots, x_n)$.

### 2.28 Definition

*Eine Funktion* $f : A \underset{p}{\longrightarrow} B$ *heißt*

1. ***surjektiv*** *oder* ***Surjektion*** *genau dann, wenn* $f(A) := \{f(a) \mid a \in A\} = B$ *ist. Ist* $f$ *total, dann sprechen wir von einer* ***Abbildung*** *von* $A$ ***auf*** $B$*. Als Relation gesehen nennt man* $f$ ***rechtstotal****.*

2. ***injektiv****,* ***eineindeutig*** *oder* ***Injektion*** *genau dann, wenn* $f^{-1}$ *eine (partielle) Funktion ist, d.h. aus* $f(x) = f(y)$ *stets* $x = y$ *folgt.*

3. ***bijektiv*** *oder* ***Bijektion*** *genau dann, wenn* $f$ *surjektiv und injektiv ist.*

### 2.29 Definition

*Seien* $A$ *und* $B$ *Mengen. Wir notieren die Menge aller Abbildungen von* $A$ *nach* $B$*, also der totalen Funktionen von* $A$ *nach* $B$*, durch* $B^A$*.*

*Ist* $A$ *zudem endlich und linear durch die Relation* $<$ *geordnet, so kann* $f : A \longrightarrow B$ *als Vektor mit* $|A|$ *Komponente, die Einträgen aus* $B$ *enthalten, betrachtet werden und wir notieren* $f \in B^A$ *(statt* $f \in B^{|A|}$*, wie häufig in der Linearen Algebra, oder statt umständlich* $f \in \underbrace{B \times B \times \cdots \times B}_{|A|-mal}$*). Die Komponenten dieses* $|A|$*-Tupels werden nun, statt mit* $1, 2, \ldots, |A|$*, mit den Elementen aus der Menge* $A$ *in aufsteigender Reihenfolge bezüglich der Ordnung* $<$ *benannt. Ist* $A = A_1 \times A_2 \times \cdots \times A_n$ *für endliche und linear geordnete Mengen* $A_i$*, so erhalten wir mit* $f \in B^A$ *eine* $|A_1| \times \cdots \times |A_n|$*-Matrix.*

Wir hatten bereits $A^n$ als das $n$-fache cartesische Produkt der Menge $A$ definiert. Alternativ könnten wir dies nun auch als die Menge aller Abbildungen $A^{\{1,2,\ldots,n\}}$ von $\{1, 2, \ldots, n\}$ auf $A$ verstehen. Mit $x \in A^n$ bezeichnet dann also der Funktionswert $x(i)$ das Element der $i$-ten Komponente des $n$-Tupels $x$. Diesen Wechsel in der Darstellungs- und Sichtweise ändern wir gerade so, wie dieser am besten passt, ähnlich wie ein Physiker Materie einmal als Teilchen und ein andermal als Welle beschreibt.

Die Notation für die Potenzmenge einer Menge $A$ mit $2^A$ (siehe Definition 2.4) wird nun noch klarer, wenn wir dies als Abkürzung für $\{0, 1\}^A$ verstehen: Die charakteristische Funktion jeder Teilmenge von $A$ ist mit dieser Schreibweise ein Element von $\{0, 1\}^A$!

Wir werden es in der Informatik, wie auch in der Mathematik, häufig mit unendlichen Mengen zu tun haben, die wir definieren und untersuchen müssen. Neben Verfahren zu ihrer Spezifikation werden wir Methoden benötigen, diese untereinander zu vergleichen und zu Klassen zusammenzustellen. Der in Definition 2.30 eingeführte Begriff der **Abzählbarkeit** beschreibt eine wichtige (notwendige) Eigenschaft unendlicher Mengen, ohne deren Erfüllung wir kein Verfahren finden könnten, mit dem wir die einzelnen Elemente der unendlichen Menge Schritt für Schritt erzeugen.

**2.30 Definition**

*Eine Menge $M$ heißt* **abzählbar** *genau dann, wenn sie entweder endlich ist oder gleich mächtig zur Menge $\mathbb{N}$ der natürlichen Zahlen.*

Für jede abzählbare Menge $M$ gilt nach dieser Definition entweder $M = \emptyset$ oder es existiert eine surjektive Abbildung $g : \mathbb{N} \longrightarrow M$ auf die Menge $M$, kurz $g(\mathbb{N}) = M$. Wir nennen $g$ dann eine **Abzählung** von $M$. Bei abzählbaren unendlichen Mengen $M$ kann immer auch eine Bijektion zwischen $\mathbb{N}$ und $M$ angegeben werden.

> Ist $M$ eine abzählbare Menge mit der Abzählung $g : \mathbb{N} \longrightarrow M$ und $g$ ist eine *berechenbare* Funktion, so heißt $M$ **aufzählbar**. Der Begriff der Berechenbarkeit einer Funktion wird zwar erst in der Vorlesung F3 definiert, es sei jedoch bereits an dieser Stelle darauf hingewiesen, dass nicht jede mathematisch wohldefinierte Funktion auch berechenbar ist! Der Grund dafür kann in der Nicht-Existenz einer endlichen Beschreibung eines endlichen Berechnungsverfahrens, also eines Algorithmus' begründet sein.

Zum Abschluss dieses Abschnittes wollen wir noch ein wichtiges Lemma beweisen.

**2.31 Lemma**

*Jede Teilmenge einer abzählbaren Menge ist auch wieder abzählbar.*

**Beweis:** Sei $f : \mathbb{N} \longrightarrow M$ eine Abzählung der Menge $M$ und $L \subseteq M$ eine beliebige Teilmenge. Gilt $L = M$ oder $L = \emptyset$, so ist nichts zu beweisen. Ist $L \neq M$, so gibt es wegen $L \neq \emptyset$ ein Element $m \in L$. Eine Abzählfunktion $g : \mathbb{N} \longrightarrow L$ wird nun definiert durch

$$g(i) := \begin{cases} f(i) & \text{falls } f(i) \in L \\ m & \text{sonst} \end{cases}.$$

□

## 2.4 Beweistechniken

In der theoretischen Informatik, wie auch in der Mathematik, werden immer wieder Standard-Beweistechniken angewandt. Zu diesen Verfahren gehören u.a. die *vollständige Induktion*, *Widerspruchsbeweise* und insbesondere *Diagonalisierungstechniken*. Im folgenden Abschnitt wollen wir einige dieser Verfahren exemplarisch diskutieren.

Auf Georg Cantor geht das mathematische Beweisverfahren der *Diagonalisierung* zurück, mit dem er beweisen konnte, dass die Menge $\mathbb{R}$ der reellen Zahlen **nicht** abzählbar ist. Da diese Technik speziell bei der Methode des indirekten Beweises auch später noch benutzt werden wird, sei sie hier an einem ersten Beispiel vorgeführt. Zuvor jedoch ein sicher bekanntes Beispiel aus der elementaren Mathematik als Anwendung und zur Verdeutlichung des Prinzips eines indirekten Beweises.

Im **indirekten Beweis** nimmt man an, die zu beweisende Aussage sei falsch. Dann benutzt man einige der Voraussetzungen (Prämissen) und andere als wahr bekannte Aussagen, die ja alle als Voraussetzungen verwendet werden können, und schließt dann, dass eine der Voraussetzungen oder eine als wahr bekannte Aussage falsch sein muss. Als Schlussfolgerung bleibt (bei richtigem Rechen) nur noch übrig, dass die zuvor als falsch angenommene Aussage wahr sein muss.

**2.32 Beispiel**

Wir wollen zeigen, dass die Aussage $s :=$ „$\frac{a+b}{2}$ ist mindestens so groß wie $\sqrt{a \cdot b}$“ gilt. Als Prämissen setzen wir voraus: $p :=$ „$a$ ist positiv reell“ und $q :=$ „$b$ ist positiv reell“. Als bekanntes Zusatzwissen nehmen wir die Aussage $r :=$ „$c^2$ ist nicht negativ für jede reelle Zahl $c$“. Die Struktur der zu beweisenden Aussage ist also: „$p$ und $q$ und $r$ implizieren $s$“, formal notiert als $p \wedge q \wedge r \rightarrow s$. Für den indirekten Beweis nehmen wir also an, dass $s$ falsch und somit $\neg s$ wahr sei. Wir nehmen also die Gültigkeit von „($\frac{a+b}{2}$ ist kleiner als $\sqrt{a \cdot b}$)“ an. Wenn wir nun einen Widerspruch ableiten können, ist Aussage $s$ gezeigt.

**Beweis (zu Beispiel 2.32):** Aus $\neg s$, formalisiert durch $(a+b) < 2 \cdot \sqrt{a \cdot b}$, folgt $(a+b)^2 < 4 \cdot a \cdot b$ und weiter $a^2 + 2 \cdot a \cdot b + b^2 < 4 \cdot a \cdot b$, d.h. $a^2 + b^2 < 2 \cdot a \cdot b$, bzw. $a^2 + b^2 - 2 \cdot a \cdot b < 0$. Dies widerspricht aber der Tatsache, dass jedes Quadrat einer reellen Zahl positiv oder Null ist (Aussage $r$). Also kann $a^2 + b^2 - 2 \cdot a \cdot b = (a-b)^2$ nach Voraussetzung $r$ nicht negativ sein und somit ist der Schluss $p \wedge q \wedge r \rightarrow s$ richtig! $\square$

Verallgemeinern wir die Aussage aus Beispiel 2.32, so erhalten wir die Aussage, dass stets das arithmetische Mittel mindestens so groß wie das geometrische Mittel ist, formal:

$$\frac{1}{n} \cdot \sum_{i=1}^{n} x_i \geq \sqrt[n]{\prod_{i=1}^{n} x_i}.$$

Das folgende Theorem wollen wir nun mittels Diagonalisierung beweisen.

**2.33 Theorem**

*Die Menge aller abzählbar unendlichen Folgen natürlicher Zahlen aus $\mathbb{N}$ ist nicht abzählbar.*

**Beweis:** Sei $F_{\text{abz}}$ die Menge aller abzählbar unendlichen Folgen natürlicher Zahlen. Wenn wir nun annehmen, dass $F_{\text{abz}}$ eine abzählbare Menge sei, so gibt es eine Abzählung, d.h., eine Abbildung $g : \mathbb{N} \longrightarrow F_{\text{abz}}$ unter der $g(i)$ für jedes $i \in \mathbb{N}$ eine abzählbar unendliche Folge natürlicher Zahlen ist. Für jede dieser Folgen $g(i)$ gibt es also eine eigene Abzählung $f_i, i \in \mathbb{N}$ der Folgenglieder. Dies können wir uns als eine unendliche Matrix vorstellen, in der in der $i$-ten Zeile die Elemente der Folge $g(i)$ von links nach rechts entsprechend ihrer Abzählung $f_i(\mathbb{N})$ eingetragen sind, wie in Abbildung 2.2:

| $i$ | $g(i)$ | 0 | 1 | 2 | 3 | 4 | 5 | $\cdots$ | $n$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $g(1)$ | $f_0(0)$ | $f_0(1)$ | $f_0(2)$ | $f_0(3)$ | $f_0(4)$ | $f_0(5)$ | $\cdots$ | $f_0(n)$ | $\cdots$ |
| 1 | $g(1)$ | $f_1(0)$ | $f_1(1)$ | $f_1(2)$ | $f_1(3)$ | $f_1(4)$ | $f_1(5)$ | $\cdots$ | $f_1(n)$ | $\cdots$ |
| 2 | $g(2)$ | $f_2(0)$ | $f_2(1)$ | $f_2(2)$ | $f_2(3)$ | $f_2(4)$ | $f_2(5)$ | $\cdots$ | $f_2(n)$ | $\cdots$ |
| 3 | $g(3)$ | $f_3(0)$ | $f_3(1)$ | $f_3(2)$ | $f_3(3)$ | $f_3(4)$ | $f_3(5)$ | $\cdots$ | $f_3(n)$ | $\cdots$ |
| 4 | $g(4)$ | $f_4(0)$ | $f_4(1)$ | $f_4(2)$ | $f_4(3)$ | $f_4(4)$ | $f_4(5)$ | $\cdots$ | $f_4(n)$ | $\cdots$ |
| 5 | $g(5)$ | $f_5(0)$ | $f_5(1)$ | $f_5(2)$ | $f_5(3)$ | $f_5(4)$ | $f_5(5)$ | $\cdots$ | $f_5(n)$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\cdots$ |
| n | $g(n)$ | $f_n(0)$ | $f_n(1)$ | $f_n(2)$ | $f_n(3)$ | $f_n(4)$ | $f_n(5)$ | $\cdots$ | $f_n(n)$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\cdots$ | $\vdots$ | $\ddots$ |

Abbildung 2.2: Angenommene Abzählung aller abzählbaren Zahlenfolgen über $\mathbb{N}$

Wenn wir nun die neue Folge $F_{\text{diagonal}}$ definieren durch

$$F_{\text{diagonal}} := \langle f_0(0)+1, f_1(1)+1, f_2(2)+1, f_3(3)+1, f_4(4)+1, \ldots\rangle, \tag{2.1}$$

so ist diese *offensichtlich* abählbar! Sie müsste also in der angenommenen Abzählung $g : I\!N \longrightarrow F_{\text{abz}}$ aller abzählbaren Folgen als $g(j)$ für ein $j \in I\!N$ vorkommen. Dann müsste jedoch definitionsgemäß, lt. Gleichung (2.1), für das Diagonalelement $f_j(j)$ gerade $f_j(j) = f_j(j) + 1$ gelten, was sicher unmöglich ist. Also kann die Annahme, $F_{\text{abz}}$ sei eine abzählbare Menge, nicht stimmen. □

Die gleiche Beweistechnik lässt sich beim Beweis des folgenden Theorems zur Abzählbarkeit der Potenzmenge zu einer gegebenen Menge anwenden.

**2.34 Theorem**
*Die Potenzmenge $2^M$ einer abzählbaren Menge $M$ ist genau dann abzählbar, wenn $M$ eine endliche Menge ist.*

**Beweis:** In Theorem 2.34 sind zwei Aussagen versteckt, die beide bewiesen werden müssen:

(a) Wenn $M$ *nicht* endlich ist, so ist die Menge $2^M$ aller Teilmengen von $M$ nicht abzählbar.

(b) Wenn $M$ endliche Menge ist, so ist $2^M$ abzählbar.

**zu (a):** Sei $M$ eine abzählbare unendliche Menge. Die surjektive Abbildung $f : I\!N \longrightarrow M$ der Abzählung der Elemente von $M$. Für jedes $i \in I\!N$ bezeichnet also $f(i)$ eine bestimmtes Element von $M$. O.B.d.A. dürfen wir annehmen, dass in dieser Abzählung von $M$ keines der Elemente doppelt auftritt, dass also $f$ sogar eine Bijektion ist! Falls nun die Potenzmenge $2^M$ ebenfalls abzählbar ist, so gibt es auch für diese Menge eine Abzählung $g : I\!N \longrightarrow 2^M$. Wir definieren die Diagonalmenge $D := \{f(j) \mid f(j) \notin g(j), j \in I\!N\}$. Offensichtlich ist $D \subseteq M$ und also $D \in 2^M$. Daher muss es eine Zahl $k \in I\!N$ mit $D = g(k)$ geben. Auf Grund der Definition der Menge $D$ bedeutet dies nun aber, dass $f(k) \in D$ genau dann gilt, wenn $f(k) \notin g(k)$, also genau dann, wenn $f(k) \notin D$ gilt. Dies ist offensichtlich ein Widerspruch und somit kann $2^M$ nicht abzählbar sein.

**zu (b):** Sei $M := \{m_0, m_1, \ldots, m_{k-1}\}$ eine endliche Menge. Wie auf Seite 13 festgestellt, hat die Potenzmenge von $M$ gerade $2^{|M|}$ viele Elemente. Somit ist auch die Menge $2^M$ endlich und damit per Definition abzählbar. □

Im vorangegangenen Beweis wurde für die endlichen Teilmengen der Menge $M$ in Teil (b) keine konkrete Abzählung konstruiert. Statt dessen haben wir ein früheres Ergebnis verwendet. Auch die Angabe einer Abzählung von $2^M$ wäre eine Möglichkeit für diesen Teilbeweis gewesen. Dies ist in unserem Fall jedoch viel aufwändiger. Der Vollständigkeit halber wollen wir diese Alternative trotzdem präsentieren.

**Beweisalternative zu Theorem 2.34 (b):**

Für $k = 0$ ist $M = \emptyset$ und auch $2^M = \emptyset$, also per Definition abzählbar. Die leere Menge ist auch in jeder Potenzmenge enthalten. Für $k \geq 1$ definieren wir nun beispielsweise die Abzählfunktion $f : I\!N \longrightarrow 2^M$ durch

$$f(x) := \begin{cases} \{m_{i_0}, \ldots, m_{i_j}\} & \text{falls } x = \sum\limits_{l=0}^{j} m_{i_l} \cdot 2^{i_l} \\ \emptyset & \text{sonst} \end{cases}$$

□

In der *konstruktiven* Beweisalternative wird eine $(k-1)$-stellige Binärzahl[7] als *Kodierung* für jede Teilmenge von $M$ verwandt. Für $0 \leq l \leq k-1$ ist die $l$-te Stelle (von rechts) der Binärdarstellung 1 genau dann, wenn das Element $m_l \in M$ in der kodierten Teilmenge enthalten ist. Zum Beispiel kodiert die Binärdarstellung 10010 gerade die Menge $\{m_1, m_4\}$. Für alle Zahlen $x$, die sich nicht auf diese Weise als Kodierung einer Teilmenge von $M$ interpretieren lassen, wird $f(x)$ die leere Menge, die ja in jeder Potenzmenge enthalten ist! Die so definierte totale Funktion $f$ ist surjektiv, aber nicht bijektiv, da jede nichtleere endliche Teilmenge von $M$ eindeutig durch ein endliches *Binärwort* aus den Werten der charakteristischen Funktion dargestellt wird, die leere Menge aber unendlich oft als Bild vorkommt.

## 2.5 Strukturen

Unter den vielen möglichen Relationen auf einer Menge sind die **Verknüpfungen** besonders hervorzuheben. Letztere sind Vorschriften, die zwei Elementen jeweils ein drittes zuordnen und werden in der Regel als **Operationen** bezeichnet. Solche (zweistelligen) Verknüpfungen können für eine Menge $M$ z.B. als Abbildungen $g : M \times M \longrightarrow M$ geschrieben werden, meistens wird für sie jedoch die Infixschreibweise verwendet. Mengen und darauf definierte Verknüpfungen sind uns von Gruppen, Ringen, Körpern oder anderen Algebren oder Strukturen bekannt.

**2.35 Definition**
*Eine Menge $A$ mit einer oder mehreren darin ausgezeichnete Relationen, Operationen und Konstanten nennt man eine* ***Struktur****. Eine Struktur wird notiert als ein Tripel $(A, F, P)$, wobei die Mengen folgende Bedeutung haben:*

$A$ *ist die nicht-leere* ***Trägermenge*** *($A \neq \emptyset$), auch* Individuenbereich *genannt, die oft mit den nullstelligen Funktionen identifiziert wird.*

$F$ *ist eine* ***Klasse von Funktionen*** *oder Abbildungen.*

$P$ *ist eine* ***Klasse von Prädikaten****, also Funktionen $f$ mit* $\mathrm{range}(f) = \{w, f\}$

*Falls in der Struktur keine Funktionen vorhanden sind, also $F = \emptyset$ ist, so heißt die Struktur* ***relationale Struktur*** *bzw.* ***Relationalstruktur*** *oder auch* ***Relationensystem****.*

*Falls $P = \emptyset$ ist, so nennt man die Struktur eine* ***algebraische Struktur*** *oder kurz* ***Algebra****.*

Die Bedingung $A \neq \emptyset$ in Defnition 2.35 ist nötig, da sonst die Aussage „$\forall x \in A : p(x)$ impliziert $\exists x \in A : p(x)$“, nicht gültig wäre.

Bisher haben wir im Wesentlichen nur relationale Strukturen behandelt, z.B., partielle Ordnungen, Äquivalenzrelationen oder Graphen. In der Informatik begegnet uns die **Halbgruppe** besonders häufig; sie ist die wohl einfachste und grundlegendste algebraische Struktur.

**2.36 Definition**
*Sei $H$ eine Menge und $\odot : H \times H \longrightarrow H$ eine Abbildung (Verknüpfung) für die das* Assoziativgesetz *gilt, d.h. $\forall a, b, c \in H : (a \odot b) \odot c = a \odot (b \odot c)$, dann heißt $(H, \odot)$* ***Halbgruppe****. Wenn klar ist, welche Verknüpfung gemeint ist, schreiben wir auch oft nur $H$ statt $(H, \odot)$.*

[7] siehe Abschnitt 2.8 für eine formale Definition des Begriff der Binärdarstellung.

**2.37 Definition**

*Eine Halbgruppe $(H, \odot)$ heißt **Monoid** genau dann, wenn es ein Element $e_H \in H$ gibt, so dass $e_H \odot m = m \odot e_H = m$ für jedes $m \in H$ gilt. Das Element $e_H \in H$ nennt man das **neutrale Element** von $(H, \odot)$. Oft wird der Index $H$ bei $e_H$ weggelassen und nur $e$ geschrieben.*

**2.38 Beispiel**

$(I\!N \setminus \{0\}, +)$ ist eine Halbgruppe, während $(I\!N, +)$ ein Monoid mit der Null als neutralem Element ist.

$(\{0, 1\}, \cdot)$ ist ein endliches Monoid mit dem neutralen Element $1$, wenn die Ziffern $0$ und $1$ als natürliche Zahlen aufgefaßt werden und die Operation $\cdot$ als gewöhnliche Multiplikation interpretiert wird.

Die Menge aller injektiven Abbildungen einer Menge $M$ in sich selbst, ist bezüglich der Hintereinanderausführung $\circ$ ein Monoid, für das die Identität $Id_M$ das neutrale Element ist. Sind $f$ und $g$ Abbildungen, so ist die neue Abbildung $f \circ g$, gesprochen „$f$ vor $g$“, erklärt durch:

$$(f \circ g)(x) = g(f(x)).$$

Dies ist äquivalent zur Komposition von Relationen. Es entspricht $(f \circ g)$ also gerade der Schreibweise $(f \cdot g)$, wenn $f$ und $g$ als Relationen aufgefaßt werden!

**2.39 Definition**

*Für Teilmengen $U, V$ einer Halbgruppe oder eines Monoides $(H, \odot)$ sei die Menge $U \cdot V := \{u \odot v \mid u \in U, v \in V\}$ das **Komplexprodukt** von $U$ und $V$.*

Mit dem Komplexprodukt zweier Teilmengen einer Halbgruppe, wird also stillschweigend die Halbgruppenoperation auf die elementweise Verknüpfung übernommen!

Damit wird die Potenzmenge von $H$ zu einer Halbgruppe $(2^H, \cdot)$, oder einem Monoid mit dem Einselement $\{e_H\}$, falls dieses in $(H, \odot)$ existiert.

**2.40 Definition**

*Eine Teilmenge $M$ von $(H, \odot)$ heißt gegenüber dem Produkt $\odot$ abgeschlossen genau dann, wenn $\forall a, b \in M : a \odot b \in M$ gilt.*

**2.41 Definition**

*Sei $(H, \odot)$ eine Halbgruppe (bzw. $(H \cup \{e_H\}, \odot)$ ein Monoid) dann heißt $M \subseteq H$ eine **Unterhalbgruppe** (bzw. **Untermonoid**) genau dann, wenn $(M, \odot)$ selbst eine Halbgruppe (bzw. Monoid) ist. Das heißt, $M$ ist gegenüber $\odot$ abgeschlossen und enthält als Untermonid auch das neutrale Element $e_H$.*

**2.42 Definition**

*Eine Abbildung $h$ von einer Halbgruppe $(H, \odot)$ auf eine andere Halbgruppe $(G, \circledast)$ wird genau dann **Homomorphismus** genannt, wenn $\forall x, y \in H : h(x \odot y) = h(x) \circledast h(y)$ gilt. Ein Homomorphismus ist also eine strukturerhaltende Abbildung, die außer den Elementen auch noch die Operationen auf einander abbildet.*

*Ein Homomorphismus $f : (H, \odot) \longrightarrow (G, \circledast)$ kurz $f : H \longrightarrow G$ heißt*

1. ***Epimorphismus** genau dann, wenn $f$ surjektiv ist*
2. ***Monomorphismus** genau dann, wenn $f$ injektiv ist*

3. ***Isomorphismus*** *genau dann, wenn $f$ bijektiv ist*

4. ***Endomorphismus*** *genau dann, wenn $H = G$ ist*

5. ***Automorphismus*** *genau dann, wenn $f$ Isomorphismus ist und $H = G$ gilt.*

Für $M \subseteq H$ definieren wir nun, ähnlich wie in Definition 2.16 für Relationen, den transitiven Abschluss unter Komplexproduktbildung.

**2.43 Definition**

*Für eine Teilmenge $M \subseteq H$ einer Halbgruppe $(H, \odot)$ (bzw. eines Monoides $(H \cup \{e_H\}, \odot)$) seien der* ***transitive Abschluss*** *$M^+$ sowie der* ***transitive und reflexive Abschluss*** *$M^*$ wie folgt erkärt:*

$$M^+ := \bigcup_{i \geq 1} M^i \text{ mit } M^1 := M \quad \text{und } M^{i+1} := M^i \cdot M$$

$$M^* := M^+ \cup \{e_H\}$$

*wobei $e_H$ das neutrale Element des Monoides ist.*

- *$M^+$ heißt* ***von $M$ erzeugte Unterhalbgruppe****.*
- *$M^*$heißt* ***von $M$ erzeugtes Untermonoid****.*
- *$M$ heißt* ***Erzeugendensystem*** *von $M^+$ bzw. von $M^*$. Ist das Erzeugendensystem $M$ endlich, so heißen $M^+$ bzw. $M^*$* ***endlich erzeugt****.*
- *$M^+$ bzw. $M^*$ heißt* ***von $M$ frei erzeugt*** *genau dann, wenn für $\forall i, k, r \in I\!N$ $\forall m_i, n_i \in M \setminus \{e_H\}$ aus $m_1 \odot m_2 \odot \ldots \odot m_k = n_1 \odot n_2 \odot \ldots \odot n_r$ stets $k = r$ und $m_i = n_i$ folgt.*

Es ist zu beachten, dass in obiger Darstellung die Mengen $M^i$ das $i$-fache Komplexprodukt bezeichnen und nicht das cartesische Produkt! Diese Bedeutung der Exponenten wird später meistens bei Wortmengen, also Teilmengen von $\Sigma^*$ für ein Alphabet $\Sigma$ verwendet. Die Definitionen hierzu finden sich in Abschnitt 2.6.

Die Darstellung eines Elementes eines von $M$ frei erzeugten Monoides $M^*$ ist stets eindeutig. Außer dem Assoziativgesetz gilt keine andere Beziehung zwischen den Erzeugenden, es ist *frei von zusätzlichen Relationen.*

Ähnlich wie in Theorem 2.26 für Relationen gezeigt, ist $M^+$ der Durchschnitt aller $M$ enthaltenden Unterhalbgruppen.

Eine von Mathematikern oft benutzte, äquivalente Definition des freien Monoides ist folgende, die auf andere Weise seine herausragende Rolle deutlich macht:

Ein Monoid $M^*$ heißt genau dann frei von $M$ erzeugt, wenn es zu jedem Homomorphismus $f : M \longrightarrow A$ für ein Monoid $A$, jeweils genau einen Homomorphismus $h : M^* \longrightarrow A$ gibt, so dass das Diagramm in Abbildung 2.3 kommutativ ist, d.h, dass $i \circ h = f$ gilt. Die Abbildung $i$ ist dabei die sogenannte natürliche Einbettung von $M$ in $M^*$ (als Teilmenge).

Hieraus folgt: Aus einer *frei wählbaren* Interpretation der Symbole eines Alphabetes $\Sigma$ in dem beliebigen Monoid $A$ (durch $f$) erhält man auf eindeutige Weise die Interpretation aller Wörter aus $\Sigma^*$.

$$\begin{tikzcd} M \arrow[rr, hook, "i"] \arrow[dr, "f"'] & & M^* \arrow[dl, "h"] \\ & A & \end{tikzcd}$$

Abbildung 2.3: Ein kommutierendes Diagramm

## 2.6 Wortmengen

In den vorangegangenen Abschnitten haben wir uns mit allgemeinen Begriffen zu Mengen, Relationen und Funktionen beschäftigt. Wir wollen nun eine spezielle Klasse von Mengen definieren: die Wortmengen. Mengen von Wörtern sind immer dort von besonderer Wichtigkeit, wenn die Objekte, mit denen gearbeitet werden soll, kodiert werden müssen. Wir hatten bereits ein solches Beispiel kennengelernt: Binärwörter. Sie können als eine Kodierung der natürlichen Zahlen angesehen werden, genau wie auch die uns geläufige Dezimaldarstellung. Allen Kodierungen gemein ist die Tatsache, dass nur eine **endliche Menge von Zeichen** zur Bildung der Darstellung erlaubt ist (Bei der Binärdarstellung nur 0 und 1, bei der Dezimaldarstellung die zehn Ziffern 0 bis 9). Eine solche Darstellung werden wir im allgemeinen als ein **Wort** über einem endlichen **Alphabet** bezeichnen.

**2.44 Definition**
*Ein **Alphabet** ist eine total geordnete endliche Menge von unterschiedlichen **Zeichen** (oder **Symbolen**), vergl. DIN 44 300.*

Besonders in der Mathematik und der Logik bezeichnet man gelegentlich auch unendliche Mengen von Symbolen als Alphabete, allerdings sind dies dann stets abzählbare, total geordnete Mengen von Zeichen. Sofern nicht ausdrücklich anders erwähnt, wollen wir unter einem Alphabet jedoch immer eine endlichen Zeichenvorrat verstehen.

Ein in der Informatik häufig verwendetes Alphabet ist der sogenannte ASCII-Code (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange), ein *7*-bit Code, dessen achtes Bit in der Byte-Darstellung als Prüfbit, oder zur Erweiterung des Zeichensatzes verwendet wird. Es ist dies eine Liste von 128 Schrift- und Steuerzeichen, die als Hexadezimalzahlen kodiert sind: In der $16 \times 8$ Matrix (siehe Tabelle 2.1) wird die Spaltennummer $j$ als Hexadezimalziffer *vor* die Hexadezimalziffer $i$ der Zeilennummer gesetzt und dann die Zeichenkette $ji$ zur Basis 16 interpretiert. Dadurch ergibt sich eine lineare Ordnung der Zeichen entsprechend ihrer Codenummern.

| |
|---|
| Einerseits kann der ASCII-Code als ein 128 Zeichen umfassendes Alphabet angesehen werden, andererseits basiert die Hexadezimaldarstellung der ASCII-Zeichen auf dem 16 zeichn umfassenden Alphabet {0, 1,2, 3, 4, 5, 6, 7, 8, 9, 0, A, B, C, D, E, F}. Diese beiden Alphabete sollten nicht verwechselt werden! |

Für das Symbol ”A“ ergibt sich gerade $[41]_{16} = 65$, für ”EOT“ (***e**nd **o**f **t**ransmission*) $[4]_{16} = 4$, für ”SP“ (***sp**ace*) $[20]_{16} = 32$ und für ”DEL“ (***del**ete*) entsprechend $[7F]_{16} = 127$. Und es ergibt sich die Reihenfolge: $\cdots \prec \mathrm{A} \prec \mathrm{B} \prec \cdots \prec \mathrm{Z} \prec \cdots \prec \mathrm{a} \prec \cdots \prec \mathrm{z} \prec \cdots$.

Tabelle 2.1: ASCII – Code

| Spalte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | NUL | DLE | SP | 0 | @ | P | ‘ | p |
| 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 2 | STX | DC2 | ” | 2 | B | R | b | r |
| 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 6 | ACK | SYN | & | 6 | F | V | f | v |
| 7 | BEL | ETB | ’ | 7 | G | W | g | w |
| 8 | BS | CAN | ( | 8 | H | X | h | x |
| 9 | HT | EM | ) | 9 | I | Y | i | y |
| A | LF | SUB | * | : | J | Z | j | z |
| B | VT | ESC | + | ; | K | [ | k | { |
| C | FF | FS | , | < | L | \ | l | \| |
| D | CR | GS | - | = | M | ] | m | } |
| E | SO | RS | . | > | N | ˆ | n | ˜ |
| F | SI | US | / | ? | O | _ | o | DEL |

In der Regel wird die Ordnung bei Alphabeten als implizit gegeben betrachtet, und jeder endliche Zeichenvorrat als Alphabet bezeichnet. Falls wir die Ordnung der Symbole bezeichnen möchten, wählen wir dazu das Zeichen $\prec$.

Um aus einem schon bekannten Wort $w$ über dem Alphabet $\{A, B, \ldots, Z\}$ ein neues zu konstruieren, kann man einzelne Symbole davor, dazwischen oder dahinter schreiben; und man kann auch zwei bekannte Wörter[8] aneinanderschreiben. Aus den Wörtern „S“, (auch ein einzelnes Symbol ist ein Wort!) „HAUS“, „BOOT“ und „BESITZER“ kann man so die Wörter BOOTS, BOOTSHAUS, HAUSBOOT, HAUSBESITZER, BOOTSBESITZER oder auch HAUSBOOTBESITZER erzeugen. Diese Hintereinanderschreibung von Zeichenketten bezeichnet man als **Konkatenation**, und mit ihr als assoziativer Operation zwischen den Zeichenketten (Wörtern) ist eine Struktur, hier eine Halbgruppe definiert. Wenn wir an einer bestimmten Stelle **kein** Wort notieren wollen, aber diese Stelle dennoch bezeichnen müssen, so benötigen wir dafür ein spezielles Symbol (ähnlich der Null bei den Zahlen). Wir nennen dieses das *leere Wort* und notieren es als $\lambda$ (*kleines lambda*). In vielen englischsprachigen Büchern findet man als Bezeichnung für das leere Wort $\epsilon$ (*epsilon*).

> Das leere Wort $\lambda$ ist niemals in einem Alphabet enthalten! Es handelt sich ja gerade um dasjenige Wort, welches ohne jegliches Symbol aus dem Alphabet gebildet wird.

**2.45 Definition**

*Für ein Alphabet $\Sigma$, ist $\Sigma^*$ das **freie Monoid** mit der **Konkatenation** oder Hintereinanderschreibung*

[8] Hier wird bewusst „Wörter“ und *nicht* „Worte“ verwendet, denn es sind Zeichenketten und keine gesprochenen Worte oder bedeutungsvolle Begriffe!

*der einzelnen Zeichen als Monoidoperation, und dem leeren Wort $\lambda$ als neutralem Element. Für $w \in \Sigma^*$ schreiben wir $w^k$ anstelle von $\underbrace{ww\cdots w}_{k}$.*

*Jede Menge $L \subseteq \Sigma^*$ heißt* ***formale Sprache****.*

$L := \{w \mid w \in \{a,b\}^* \wedge (w$ enthält das Symbol $a$ doppelt so oft wie das Symbol $b)\}$ ist zum Beispiel eine recht einfache formale Sprache mit umständlicher Beschreibung. Auch die Menge $\{w \mid w \in \{a,b,c\}^* \wedge (w$ beginnt mit dem Symbol $a)\}$ ist eine einfache formale Sprache mit unendlich vielen Elementen.

Einige Eigenschaften von Zeichenketten werden wir bei der Beschreibung und Spezifikation von formalen Sprachen benötigen, um solche umständlichen Definitionen vermeiden zu können:

**2.46 Definition**

- *Zu jedem Wort $w \in \Sigma^*$ und jedem Symbol $x \in \Sigma$ bezeichne $|w|_x$ die* ***Häufigkeit des Vorkommens*** *des Symbols $x$ im Wort $w$.*
- *Mit $|w|$ wird die* ***Länge des Wortes*** *$w$ bezeichnet, d.h. $|w| := \sum_{x\in\Sigma} |w|_x$.*
- *Wir schreiben $u \sqsubseteq v$, wenn es ein Wort $w \in \Sigma^*$ gibt, so dass $v = uw$ ist. Das Wort $u$ wird* ***Präfix*** *von $v$ genannt.*
- *Das Wort $u$ wird* ***echtes Präfix*** *von $v$ genannt, wenn $u \sqsubseteq v$ und weder $u = v$ noch $u = \lambda$ gilt. Wir notieren dies als $u \sqsubset v$.*
- *Gilt $z = uvw$ für Wörter $u, v, w, z \in \Sigma^*$, so ist $v$ ein* ***Teilwort*** *von $z$.*

**2.47 Beispiel**

Für jedes Wort $w$ gilt $\lambda \sqsubseteq w$. Ebenso ist das leere Wort Teilwort eines jeden beliebigen Wortes!

Sei $\Sigma := \{L, R\}$, dann ist für $w := LLRLRLLR$ gerade $|w| = 8$, $|w|_L = 5$, und $|w|_R = 3$.

Sei $D_1 := \{w \in \Sigma^* \mid |w|_L = |w|_R$ und $\forall u \in \Sigma^* : (u \sqsubseteq w) \rightarrow (|u|_L \geq |u|_R)\}$. Diese (formale) Sprache wird (*einseitige*) *Dyck-Sprache* über einem Klammepaar genannt, mit dem Symbol $L$ als öffnender und $R$ als schließender Klammer.

$(D_1, \cdot)$ ist selbst ein Monoid, weil auch das leere Wort $\lambda$ dazugehört und mit $u, v \in D_1$, stets auch $w := u \cdot v = uv \in D_1$ gilt, d.h. $D_1^* = D_1$ ist. Dieses Untermonoid kann jedoch niemals von einer endlichen Menge frei erzeugt werden! Den Beweis für diese Tatsache bleiben wir hier jedoch schuldig.

Betrachten wir $M := \{L, LR, RL\} \subsetneq \Sigma^*$ und bilden $M^*$, so ist auch $M^*$ ein Untermonoid von $\Sigma^*$, jedoch ist dieses von $M$ nicht frei erzeugt. Das Element $LRL \in M^*$ hat zwei verschiedene Zerlegungen in einzelne Erzeugende: $LRL = L \cdot RL = LR \cdot L$.

Neben der Konkatenation, benutzen wir eine große Zahl weiterer Operationen auf Wörtern, von denen wir hier nicht alle sofort ohne ihren Zusammenhang zu möglichen „An- und Verwendungen“ definieren wollen. Zu den wichtigen gehören zum Beispiel die folgende, die wir als weiteres Beispiel einer rekursiven Definition angeben.

**2.48 Definition**
*Sei $\Sigma$ ein Alphabet, dann ist für jedes Wort $w \in \Sigma^*$ das* ***Spiegelwort*** *$w^{\text{rev}}$ definiert durch:*

$$(ux)^{\text{rev}} := xu^{\text{rev}} \quad \textit{für} \quad x \in \Sigma, u \in \Sigma^*, \quad \textit{und} \quad \lambda^{\text{rev}} := \lambda.$$

*Für $L \subseteq \Sigma^*$ wird die Spiegelwortabbildung erweitert durch: $L^{\text{rev}} := \{w^{\text{rev}} \mid w \in L\}$.*

**2.49 Beispiel**

- $\text{REGAL}^{\text{rev}} = \text{LAGER}$,
- $\text{MARKTKRAM}^{\text{rev}} = \text{MARKTKRAM}$ und
- $\text{RENTNER}^{\text{rev}} = \text{RENTNER}$.

Ein Wort, für das $w^{\text{rev}} = w$ gilt, nennt man **Palindrom**.

Aus der rekursiven Definition der Spiegelwörter ergeben sich auch hier wieder induktive Beweise. Für das folgende einfache Lemma soll der letzte ausführliche Beweis dieser Art vorgeführt werden.

**2.50 Lemma**
*Für jedes Alphabet $\Sigma$ gilt: $\forall u, v \in \Sigma^* : (uv)^{\text{rev}} = v^{\text{rev}}u^{\text{rev}}$.*

**Beweis:** Wir verwenden vollständige Induktion über die Länge von $v$:

**Induktions-Basis:** $|v| = 0$, also ist $v = \lambda$ und $(uv)^{\text{rev}} = (u\lambda)^{\text{rev}} = u^{\text{rev}} = \lambda u^{\text{rev}} = \lambda^{\text{rev}}u^{\text{rev}} = v^{\text{rev}}u^{\text{rev}}$.

**Induktions-Annahme:** $|v| = n$, impliziert $(uv)^{\text{rev}} = v^{\text{rev}}u^{\text{rev}}$.

**Induktions-Schritt:** Sei $|v| = n + 1$, dann ist $v = wx$ für ein $x \in \Sigma$ und es gilt $w \in \Sigma^*$ mit $|w| = n$.

$$\begin{aligned}
(uv)^{\text{rev}} &= (u(wx))^{\text{rev}} && \text{da } v = wx \\
&= ((uw)x)^{\text{rev}} && \text{Konkatenation ist assoziativ} \\
&= x(uw)^{\text{rev}} && \text{Definition von } ((uw)x)^{\text{rev}} \\
&= x(w)^{\text{rev}}(u)^{\text{rev}} && \text{Induktionsannahme} \\
&= (wx)^{\text{rev}}(u)^{\text{rev}} && \text{Definition von } (wx)^{\text{rev}} \\
&= (v)^{\text{rev}}(u)^{\text{rev}} && \text{da } v = wx.
\end{aligned}$$

□

Wir wollen nun einige Ordnungen auf Wortmengen definieren. Wir betrachten zunächst ein Beispiel:

**2.51 Beispiel**
Sei $w_1 \ll w_2$ für Wörter $w_1, w_2 \in \Sigma^*$ genau dann, wenn $|w_1| \leq |w_2|$ ist. Offensichtlich ist $(\Sigma^*, \ll)$ keine totale Ordnung, denn Wörter gleicher Länge lassen sich mit $\ll$ nicht vergleichen!

Um alle Wörter untereinander vergleichbar zu machen, kann man jedoch die sogenannte *lexikographische Ordnung* als Erweiterung der Ordnung $\prec$ auf dem zugrundeliegenden Alphabe, also $(\Sigma, \prec)$, betrachten. Wir definieren diese Ordnung rekursiv:

**2.52 Definition**

*Sei $(\Sigma, \prec)$ ein mit $\prec$ linear geordnetes Alphabet, dann ist die* ***lexikographische Erweiterung*** $\prec^{\text{lex}} \subseteq \Sigma^* \times \Sigma^*$ *von $\prec$ definiert durch:*

1. $\lambda \prec^{\text{lex}} w$ *für alle* $w \in \Sigma^*$.
2. $\forall\, x, y \in \Sigma\ \forall\, u, v \in \Sigma^+ : (xu \prec^{\text{lex}} yv)$ *genau dann, wenn* $\begin{cases} (x \prec y) \text{ oder} \\ (x = y) \text{ und } (u \prec^{\text{lex}} v) \end{cases}$

$\preceq^{\text{lex}} := \prec^{\text{lex}} \cup Id_{\Sigma^*}$ *nennen wir* ***lexikographische Ordnung*** *auf* $\Sigma^*$.

Gestützt auf diese rekursive Definition lassen sich nun einige Eigenschaften leicht induktiv beweisen.

**2.53 Lemma**

*Sei $(\Sigma, \prec)$ ein linear geordnetes Alphabet, dann gilt:*

*a)* $\forall v \in \Sigma^*, w \in \Sigma^+ : v \prec^{\text{lex}} vw$

*b)* $\forall u, v \in \Sigma^* : (u \preceq^{\text{lex}} v)$ *impliziert* $(u \sqsubseteq v)$ *oder* $\big(\forall w_1, w_2 \in \Sigma^* : uw_1 \prec^{\text{lex}} vw_2\big)$

**Beweis:**

**zu a):** Wir beweisen $(*)$: $v \prec^{\text{lex}} vw$ für $w \neq \lambda$ durch vollständige Induktion über die Länge des Präfixes $v$.

**Induktions-Basis:** Für $|v| = 0$ ist $v = \lambda$ und $(*)$ gilt nach *1.* der Definition.

**Induktions-Annahme:** Wir nehmen an, dass $(*)$ für alle $v$ mit $|v| = k$ gilt.

**Induktions-Schritt:** Jedes $v \in \Sigma^*$ mit $|v| = k+1$ hat die Form $v = xu$ für ein $x \in \Sigma$. Nach *2.* der Definition von $\prec^{\text{lex}}$ gilt aber $xu \prec^{\text{lex}} xuw$ immer dann, wenn $u \prec^{\text{lex}} uw$ zutrifft. Dies gilt mit $|u| = k$ aber wegen der Induktionsannahme. Folglich gilt $(*)$ nun für alle $v \in \Sigma^*$.

**zu b):** Nach a) ist die Implikation „$u \preceq^{\text{lex}} v$ impliziert $u \sqsubseteq v$“ richtig, wenn $u$ tatsächlich ein Präfix von $v$ ist. Sollte das nicht der Fall sein, so gibt es einen gemeinsamen (eventuell den trivialen: $\lambda$) Präfix $w \sqsubseteq u$ und $w \sqsubseteq v$ und zwei Symbole $x, y \in \Sigma$ mit $wx \sqsubset u$, $wy \sqsubset v$ und $x \prec y$, so dass $wx \prec^{\text{lex}} wy$. Entsprechend 3. der Definition kommt es dann nicht mehr darauf an, wie die echten Präfixe $wx$ und $wy$ von $u$ und $v$ weiter nach rechts verlängert werden, so dass $\forall w_1, w_2 \in \Sigma^* : uw_1 \prec^{\text{lex}} vw_2$ gilt. Die Implikation ist daher gültig.

□

**2.54 Beispiel**

Es gilt unter der üblichen linearen Ordnung der Symbole des lateinischen Alphabetes folgende lexikographische Anordnung:

$$\mathsf{AB} \prec^{\text{lex}} \mathsf{FBI} \prec^{\text{lex}} \mathsf{KLAUSUR} \prec^{\text{lex}} \mathsf{OB} \prec^{\text{lex}} \mathsf{PI} \prec^{\text{lex}} \mathsf{UNI} \prec^{\text{lex}} \mathsf{ZOO}$$

In der deutschen Sprache ist die dem Alphabet unterliegende Ordnung $\prec$ bezüglich der Umlaute nicht eindeutig definiert: Im Duden wird „ä“ wie „a“ behandelt, während es im Telefonbuch wie „ae“ einsortiert wird. In keinem Fall sind die Umlaute in die totale Ordnung $\prec$ des Alphabets integriert!

Die lexikographische Striktordnung $\prec^{\text{lex}}$ ist total, und besitzt kein Supremum und keine maximalen Elemente! Die Kette $a \prec^{\text{lex}} aa \prec^{\text{lex}} aaa \prec^{\text{lex}} \cdots$ kann unendlich fortgesetzt werden und jedes Element der ebenfalls unendlichen Kette $b \prec^{\text{lex}} bb \prec^{\text{lex}} bbb \prec^{\text{lex}} \cdots$ ist bezüglich $\prec^{\text{lex}}$ größer als jedes beliebige Element $a^i$ der ersten unendlichen, aufsteigenden Kette. Die in Definition 2.52 eingeführte Relation $\ll$ eignet sich nicht als Grundlage für eine Abzählung aller Wörter, sofern das zugrundeliegende Alphabet mindestens zwei Symbole enthält. Dann ist es nämlich nicht möglich, ein Verfahren, d.h. eine Abbildung $g : I\!N \longrightarrow \Sigma^*$, anzugeben, mit dem die Wörter aus $\Sigma^*$, mit $|\Sigma| \geq 2$, so abgezählt werden, dass stets $g(i) \prec^{\text{lex}} g(i+1)$ gilt! Um das zu erreichen wird in der Regel eine andere Ordnung auf $\Sigma^*$ definiert, die wir **lexikalisch** nennen. In ihr werden die Wörter zuerst nach ihrer Länge verglichen (kürzere Wörter sind btgl. der lexikalischen Ordnung kleiner als längere) und nur Wörter gleicher Länge werden noch lexikographisch geordnet.

**2.55 Definition**

*Sei $(\Sigma, \prec)$ ein mit $\prec$ linear geordnetes Alphabet, dann ist die* ***lexikalische*** *Erweiterung $\prec^{\text{lg}-\text{lex}}$ von $\prec$ definiert durch:*

*$w_1 \prec^{\text{lg}-\text{lex}} w_2$ gelte für beliebige Wörter $w_1, w_2 \in \Sigma^*$ genau dann, wenn*

$$\text{entweder: } \quad |w_1| < |w_2|, \qquad \text{oder: } \quad |w_1| = |w_2| \quad \wedge \quad w_1 \prec^{\text{lex}} w_2.$$

*$\preceq^{\text{lg}-\text{lex}} := \prec^{\text{lg}-\text{lex}} \cup Id_{\Sigma^*}$ nennen wir kurz lexikalische Ordnung auf $\Sigma^*$.*

**2.56 Beispiel**

Es gilt unter der üblichen linearen Ordnung der Symbole des lateinischen Alphabetes folgende lexikalische Anordnung:

$$\mathsf{AB} \prec^{\text{lg}-\text{lex}} \mathsf{OB} \prec^{\text{lg}-\text{lex}} \mathsf{PI} \prec^{\text{lg}-\text{lex}} \mathsf{FBI} \prec^{\text{lg}-\text{lex}} \mathsf{UNI} \prec^{\text{lg}-\text{lex}} \mathsf{ZOO} \prec^{\text{lg}-\text{lex}} \mathsf{KLAUSUR}$$

Mit dieser Definition ist es nun ein Leichtes, ein Verfahren zur sukzessiven Erzeugung aller Wörter von $\Sigma^*$ für eiin beliebiges aber festes Alphabet $\Sigma$ zu definieren: In Schritt (1) beginnt man mit $k := 0$, notiert das kleinste Wort $\lambda$ und geht zu Schritt (2). In Schritt (2) erhöht man $k$ um eins und geht zu Schritt (3). In Schritt (3) notiert man die $|\Sigma|^k$ unterschiedlichen Wörter der Länge $k$ in lexikographisch aufsteigender Rehenfolge. Dann fährt man wieder mit Schritt (2) fort.

Weniger leicht ist es vielleicht, eine bijektive Abzählfunktion zu formulieren, die auf Grund der Existenz des eindeutigen Erzeugungsverfahrens ja existieren muss: Zur gegebenen Zahl $i \in I\!N$ läßt man das angegebene Verfahren (beginnend mit dem minimalen Wort bezüglich $\prec^{\text{lg}-\text{lex}}$) solange laufen, bis das $i$-te Wort erzeugt wurde. Den eindeutig bestimmten Index $i$ eines beliebigen Wortes kann man umgekehrt mit dem Verfahren der Abzählung ebenfalls gewinnen: Man erzeugt sukzessive die Wörter mit dem Verfahren solange, bis das vorliegende Wort erreicht wird. Die Anzahl der erzeugten Wörter muss dabei nur mitgezählt werden, um den Index zu ermitteln. Wir werden später noch ein besseres Verfahren kennenlernen.

**Verallgemeinerung auf Tupel von reellen Zahlen** Oftmals benötigt man Abzählungen und Ordnungen auf $m$-Tupeln (Vektoren) aus $\mathbb{R}^m$. Ähnlich zur lexikographischen (und auch zur lexikalischen) Ordnung auf $\Sigma^*$ definiert man Ordnungen $<^{\text{lex}} \subseteq \mathbb{R}^m$ (bzw. $<^{\text{lg-lex}} \subseteq \mathbb{R}^m$) für Tupel von reellen Zahlen:

**2.57 Definition**
*Sei $m \in \mathbb{Z}^+$ dann ist die **lexikographische Erweiterung** $<^{\text{lex}} \subseteq \mathbb{R}^m$ von $< \subseteq \mathbb{R}$ definiert durch:*

*$v_1 <^{\text{lex}} v_2$ gelte für beliebige $v_1, v_2 \in \mathbb{R}^m$ genau dann, wenn*

$$\exists k \in \{1, \ldots, m\} : (\forall i : (1 \le i < k) \to (v_1(i) = v_2(i)) \wedge (v_1(k) < v_2(k)).$$

*$\le^{\text{lex}} := <^{\text{lex}} \cup Id_{\mathbb{R}^m}$ nennen wir **lexikographische Ordnung** auf $\mathbb{R}^m$.*

Darauf aufbauend auch für Vektoren die entsprechende lexikalische Variante:

**2.58 Definition**
*Sei $m \in \mathbb{Z}^+$ dann ist die **lexikalische Erweiterung** $<^{\text{lg-lex}} \subseteq \mathbb{R}^m$ von $< \subseteq \mathbb{R}$ definiert durch:*

*$v_1 <^{\text{lg-lex}} v_2$ gelte für beliebige $v_1, v_2 \in \mathbb{R}^m$ genau dann, wenn*

$$\text{entweder:} \quad \sum_{i=1}^{m} v_1(i) < \sum_{i=1}^{m} v_2(i) \qquad \text{oder:} \quad (\sum_{i=1}^{m} v_1(i) = \sum_{i=1}^{m} v_2(i)) \wedge (v_1 <^{\text{lex}} v_2).$$

*$\le^{\text{lg-lex}} := <^{\text{lg-lex}} \cup Id_{\mathbb{R}^m}$ nennen wir **lexikalische Ordnung** auf $\mathbb{R}^m$.*

In beiden Fällen, wie sonst auch üblich, gilt Gleichheit von $m$-Tupeln nur bei komponentenweiser Gleichheit! Auch sonst ist es gebräuchlich, eine auf $\mathbb{R}$ definierte Relationen $\ll \subseteq \mathbb{R}$ so auf $\mathbb{R}^m$ zu übertragen, dass $v_1 \ll v_2$ für beliebige $v_1, v_2 \in \mathbb{R}^m$ genau dann gilt, wenn $v_1(i) \ll v_2(i)$ für jede Komponente $i \in \{1, \ldots, m\}$ zutrifft. Damit wird jede partielle Ordnung auf $\ll \subseteq \mathbb{R}$ zu einer partiellen Ordnung auf $\mathbb{R}^m$. Falls $\ll \subseteq \mathbb{R}$ totale Ordnung ist, so ist deren komponentenweise Erweiterung auf $\mathbb{R}^m$ in der Regel nur noch partiell und ausschließlich im Falle $m = 1$ (trivialer Weise) ebenfalls total.

Für die Wörter der Form $a^i b^j$ gilt bei $a \prec b$ die Beziehung $a^i b^j \prec^{\text{lg-lex}} a^r b^s$ genau dann, wenn $(r, s) <^{\text{lg-lex}} (i, j)$ zutrifft.

Die Tupel $(i, j) \in \mathbb{N}^2$ in der zweidimensionalen, unendlichen Aufstellung nach Tabelle 2.2 werden, wie in Tabelle 2.3 angedeutet, entsprechend der Ordnung $<^{\text{lg-lex}}$ abgezählt.

## 2.7 Paar- und Tupelfunktionen

Häufig kommt es vor, dass man eine Abzählung oder Nummerierung einer (unendlichen) Objektmenge $M$ benötigt, z.B. um eine systematische Betrachtung *aller* Objekte zu ermöglichen.[9] **Paarfunktionen** sind Spezialfälle von Nummerierungen, nämlich Nummerierungen von Paaren aus $\mathbb{N}^2$. Sie können auf einfache Weise durch geeignete Schachtelung zu sogenannten **Tupelfunktionen** erweitert werden. Somit brauchen wir anstelle aller Funktionen mit Vorbereich $\mathbb{N}^m$ und Nachbereich $\mathbb{N}^n$ nur noch einfache

[9] Eine Nummerierung ist immer eine surjektive Abbildung der Form $\mathbb{N} \longrightarrow M$.

Tabelle 2.2: Tabelle der Tupel aus $I\!N \times I\!N$

| j<br>i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $(0,0)$ | $(0,1)$ | $(0,2)$ | $(0,3)$ | $(0,4)$ | $(0,5)$ | $(0,6)$ | $(0,7)$ | (0,8) | $\cdots$ |
| 1 | $(1,0)$ | $(1,1)$ | $(1,2)$ | $(1,3)$ | $(1,4)$ | $(1,5)$ | $(1,6)$ | $(1,7)$ | (1,8) | $\cdots$ |
| 2 | $(2,0)$ | $(2,1)$ | $(2,2)$ | $(2,3)$ | $(2,4)$ | $(2,5)$ | $(2,6)$ | $(2,7)$ | (2,8) | $\cdots$ |
| 3 | $(3,0)$ | $(3,1)$ | $(3,2)$ | $(3,3)$ | $(3,4)$ | $(3,5)$ | $(3,6)$ | $(3,7)$ | (3,8) | $\cdots$ |
| 4 | $(4,0)$ | $(4,1)$ | $(4,2)$ | $(4,3)$ | $(4,4)$ | $(4,5)$ | $(4,6)$ | $(4,7)$ | (4,8) | $\cdots$ |
| 5 | $(5,0)$ | $(5,1)$ | $(5,2)$ | $(5,3)$ | $(5,4)$ | $(5,5)$ | $(5,6)$ | $(5,7)$ | (5,8) | $\cdots$ |
| 6 | $(6,0)$ | $(6,1)$ | $(6,2)$ | $(6,3)$ | $(6,4)$ | $(6,5)$ | $(6,6)$ | $(6,7)$ | (6,8) | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ |

Zahlenfunktionen der Art $f : I\!N \longrightarrow I\!N$ zu betrachten, da alle anderen Funktionen durch Komposition mit entsprechenden Tupelfunktionen konstruiert werden können.[10]

Natürlich ist es leicht, ein Verfahren zum sukzessiven Erzeugen des jeweils nächst größeren Tupels in $I\!N^2$ aufzuschreiben. Weit schwieriger ist es jedoch, eine wirkliche Bijektion zwischen $I\!N$ und $I\!N^2$ mit geschlossenen Formeln anzugeben.

Auf den Begründer der Mengenlehre, Georg Cantor (1845 – 1918), geht das im Folgenden beschriebene Verfahren mit der Cantorschen Paarfunktion paar : $I\!N \times I\!N \longrightarrow I\!N$ zurück. Sie erzeugt gerade die Abzählung aus Tabelle 2.3. Diese Funktion ist jedoch nicht die einzige Art, eine Bijektion zwischen den Mengen $I\!N^2$ und $I\!N$ anzugeben. Im Allgemeinen ist eine Funktion zu finden, welche ähnliche Eigenschaften, wie die spezielle Paarfunktion paar hat. Die benötigten Eigenschaften werden in der folgenden Definition fetsgelegt.

Tabelle 2.3: Abzählung der Tupel aus $I\!N \times I\!N$ entsprechend $<^{\mathrm{lg-lex}}$:

| j<br>i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 3 | 6 | 10 | 15 | 21 | 28 | $\cdots$ |
| 1 | 2 | 4 | 7 | 11 | 16 | 22 | 29 | $\cdots$ | |
| 2 | 5 | 8 | 12 | 17 | 23 | 30 | $\cdots$ | | |
| 3 | 9 | 13 | 18 | 24 | 31 | $\cdots$ | | | |
| 4 | 14 | 19 | 25 | 32 | $\cdots$ | | | | |
| 5 | 20 | 26 | 33 | $\cdots$ | | | | | |
| 6 | 27 | 34 | $\cdots$ | | | | | | |
| 7 | 35 | $\cdots$ | | | | | | | |
| $\vdots$ | $\cdots$ | | | | | | | | |

**2.59 Definition**

*Eine Funktion* pair : $I\!N \times I\!N \longrightarrow I\!N$, *nennt man* ***Paarfunktion*** *(pairing funktion), wenn:*

[10] Die hierzu notwendigen Eigenschaften von Tupelfunktionen, wie die Berechenbarkeit, werden wir hier einfach voraussetzen. Sie werden Gegenstand der Vorlesung F3 sein.

1. pair *surjektiv ist und* $\mathrm{pair}(i,j)$ *effektiv bestimmbar ist, und*

2. *zwei gleichfalls berechenbare Funktionen* $\pi_1 : \mathbb{N} \longrightarrow \mathbb{N}$ *und* $\pi_2 : \mathbb{N} \longrightarrow \mathbb{N}$ *existieren, für die:* $\pi_1(\mathrm{pair}(i,j)) = i$, $\pi_2(pair(i,j)) = j$ *und* $\mathrm{pair}(\pi_1(k), \pi_2(k)) = k$ *gilt.*

Betrachten wir die Ordnung $<^{\mathrm{lg-lex}} \subseteq \mathbb{N}^2$ und die Tupel aus $\mathbb{N}^2$ in aufsteigender Reihenfolge bezüglich $<^{\mathrm{lg-lex}}$. Wie in Abbildung 2.3 werden wir an der Stelle $(i,j)$, also am Schnittpunkt von $i$-ter Zeile und $j$-ter Spalte der nach unten offenen Anordnung, genau die Zahl $\mathrm{paar}(i,j)$ eintragen, die bei Abzählung der Tupel (beginnend bei 0 (*Null*) für das Tupel $(0,0)$) entsprechend dieser Ordnung entsteht.

Für das $k$-te Tupel $(i,j)$ gilt die Gleichung:

$$k = \sum_{r=0}^{i+j} + i$$

Die Gültigkeit dieser Gleichung kann folgendermaßen gezeigt werden:

Bezeichnet man die Nebendiagonale (*counter diagonal*) in der $k := \mathrm{paar}(i,j)$ steht mit $cd(k)$, so gilt offensichtlich $cd(\mathrm{paar}(i,j)) = i + j$. In der Nebendiagonalen mit der Nummer $n$ findet man stets genau $n+1$ Elemente, von denen jedes genau um 1 größer ist als das Vorangegangene. Daher ist $\mathrm{paar}(0,j) = \sum_{r=0}^{j} r$, und für dieses und die weiteren Elemente $0 \leq t \leq i$ derselben Nebendiagonalen $j$ gilt dann: $\mathrm{paar}(t, j-t) = \sum_{r=0}^{j} r + t$. Also ist die Funktion $\mathrm{paar} : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, definiert durch

$$\mathrm{paar}(i,j) := \frac{(i+j)(i+j+1)}{2} + i$$

ein Kandidat für eine Paarfunktion, wenn wir auch noch ihre Umkehrung mit Hilfe der Funktionen $\pi_1$ und $\pi_2$ angeben können.

Hierbei ist natürlich eine geschlossene Form erwünscht und nicht nur ein rekursives Bestimmungs- oder iteratives Erzeugungsverfahren (vergl. [Gruska])! Wie schon bemerkt, ist die Komponentensumme $\pi_1(\mathrm{paar}(i,j)) + \pi_2(\mathrm{paar}(i,j)) = i + j$ auf den Nebendiagonalen stets konstant so groß wie der maximale Zeilen- oder Spaltenindex dieser Nebendiagonalen. Wenn nun eine Zahl $z := \mathrm{paar}(i,j)$ gegeben ist, aber weder die Nummer $cd(z)$ der Nebendiagonalen zu $z$, noch die Größen $i$ oder $j$, so wäre $\mathrm{paar}(0, i+j)$ die kleinste Zahl in dieser Nebendiagonalen $cd(z)$. Es ist dies die Zahl

$$k := \mathrm{paar}(0, i+j) = \frac{(i+j)(i+j+1)}{2} = \frac{cd(k)(cd(k)+1)}{2}.$$

Durch Auflösen nach $cd(k)$ erhält man:

$$\begin{aligned}
\mathrm{paar}(0, i+j) &= \frac{cd(k)(cd(k)+1)}{2}, \\
2 \cdot \mathrm{paar}(0, i+j) &= cd(k)(cd(k)+1), \\
2 \cdot \mathrm{paar}(0, i+j) + \frac{1}{4} &= (cd(k))^2 + cd(k) + \frac{1}{4}
\end{aligned}$$

$$
\begin{aligned}
&= (cd(k) + \frac{1}{2})^2, \\
\sqrt{2 \cdot \text{paar}(0, i+j) + \frac{1}{4}} - \frac{1}{2} &= cd(k).
\end{aligned}
$$

Es ist nun die Abbildung paar monoton, d.h.:

$$(i_1, j_1) <^{\text{lg}-\text{lex}} (i_2, j_2) \quad \text{impliziert} \quad \text{paar}(i_1, j_1) < \text{paar}(i_2, j_2).$$

Ebenfalls monoton ist die Abbildung, die $k \in I\!N$ den Wert $\sqrt{2k + \frac{1}{4}} - \frac{1}{2}$ zuordnet. Man rechnet leicht nach, dass

$$\text{paar}(i+j, 0) + 1 = \text{paar}(0, i+j+1)$$

die kleinste Zahl auf der nächst höheren Nebendiagonale ist, so dass (für $i, j \neq 0$):

$$
\begin{aligned}
cd(\text{paar}(0, i+j)) &= \sqrt{2 \cdot \text{paar}(0, i+j) + \frac{1}{4}} - \frac{1}{2} \\
&< \sqrt{2 \cdot \text{paar}(i, j) + \frac{1}{4}} - \frac{1}{2} \\
&< \sqrt{2 \cdot \text{paar}(i+j, 0) + \frac{1}{4}} - \frac{1}{2} \\
&< \sqrt{2 \cdot (\text{paar}(i+j, 0) + 1) + \frac{1}{4}} - \frac{1}{2} \\
&= \sqrt{2 \cdot \text{paar}(0, i+j+1) + \frac{1}{4}} - \frac{1}{2} \\
&= cd(\text{paar}(0, i+j+1)) \\
&= cd(\text{paar}(i, j)) + 1
\end{aligned}
$$

Daher ergibt sich eine geschlossene Formel zur Berechnung der Nebendiagonalen in der sich eine gegeben Zahl $k$ befindet:

$$cd(k) = \left\lfloor \sqrt{2k + \frac{1}{4}} - \frac{1}{2} \right\rfloor.$$

Daraus kann über $k = \text{paar}(\pi_1(k), \pi_2(k)) = \frac{cd(k)(cd(k)+1)}{2} + \pi_1(k)$ zunächst $\pi_1(k)$, und aus $cd(k) = \pi_1(k) + \pi_2(k)$ sofort $\pi_2(k)$ ermittelt werden:

$$
\begin{aligned}
\pi_1(k) &= k - \frac{cd(k)(cd(k)+1)}{2} \\
&= k - \frac{1}{2} \cdot \left\lfloor \sqrt{2 \cdot \text{paar}(i, j) + \frac{1}{4}} - \frac{1}{2} \right\rfloor \cdot \left\lfloor \sqrt{2 \cdot \text{paar}(i, j) + \frac{1}{4}} + \frac{1}{2} \right\rfloor \\
\pi_2(k) &= cd(k) - \pi_1(k)
\end{aligned}
$$

Mit einer Paarfunktion pair : $I\!N^2 \longrightarrow I\!N$ kann man auch **Tupelfunktionen** $\pi^{(k)}$ für höhere Dimensionen $k \in I\!N$ erklären:

**2.60 Definition**

*Die $k$-**Tupelfunktion** $\pi^{(k)} : \mathbb{N}^k \longrightarrow \mathbb{N}$ für $k > 2$ wird zurückgeführt auf die übliche Paarfunktion* pair *durch:*

$$\pi^{(k)}(x_1, x_2, \ldots, x_k) := \mathrm{pair}(x_1, \pi^{(k-1)}(x_2, \ldots, x_k)) = \underbrace{\mathrm{pair}(x_1, \ldots, \mathrm{pair}(x_{k-1}, x_k)}_{(k-1)\text{-mal}} \ldots)$$

Die zu einer $k$-Tupelfunktione zugehörigen Projektionen auf die einzelnen Komponenten, die sogenannten **$k$-Komponentenfunktionen**, werden mit $\pi_i^{(k)}$ bezeichnet und sind für $i \in \{1, \ldots, k\}$ ebenfalls bestimmbar.

**2.61 Beispiel**

Sei $\vec{x} := (2, 5, 7, 3)$ ein Quadrupel (4-Tupel) aus $\mathbb{N}^4$. Mit $\pi^{(4)}(\vec{x})$ wird die natürliche Zahl

$$z := \mathrm{pair}(2, \mathrm{pair}(5, \mathrm{pair}(7, 3)))$$

bezeichnet. Die dritte Komponente des ursprünglichen Tupels lässt sich nun mit der Komponentenfunktion $\pi_3^{(4)}$ zurückgewinnen:

$$\pi_3^{(4)}(z) = \pi_3^{(4)}(\pi^{(4)}(2, 5, 7, 3)) = 7$$

## 2.8 Nummerierungen von Wortmengen

Auf Kurt Gödel geht die Idee zurück, beliebige Zeichenketten durch natürliche Zahlen zu kodieren. Die Nummerierungen allein der Buchstaben, wie zum Beispiel beim ASCII-Code, geben ja noch keine Vorschrift zum Abzählen aller Wörter.

**Gödelisierung** Solch eine Zuordnung von Zahlen zu Wörtern nützt in vielen Fällen nur dann etwas, wenn das Ursprungsobjekt der Kodierung aus dieser auch wieder hergestellt werden kann. Daher wird die Gödelnummerierung der Wortmenge über einem Alphabet als injektive Funktion definiert.

Wir verwenden für diese Nummerierungen das griechische Symbol $\nu$ (sprich: nü) mit einem Index, der die jeweils gemeinte Nummerierung bezeichnet.

**2.62 Definition**

*Sei $\Sigma$ ein endliches Alphabet. Eine Funktion $\nu_\Sigma : \Sigma^* \longrightarrow \mathbb{N}$ heißt **Gödelisierung**, wenn gilt:*

1. *$\forall u, v \in \Sigma^* : u \neq v$ impliziert $\nu_\Sigma(u) \neq \nu_\Sigma(v)$, $\nu_\Sigma$ ist also injektiv.*
2. *Aus jedem Argument $w \in \Sigma^*$ kann $\nu_\Sigma(w)$ in endlich vielen Schritten effektiv bestimmt werden.*
3. *Für jedes $n \in \mathbb{N}$ kann in endlich vielen Schritten effektiv bestimmt werden, ob es ein $w \in \Sigma^*$ gibt, für das $\nu_\Sigma(w) = n$ gilt.*
4. *Wenn es ein $w \in \Sigma^*$ mit $\nu_\Sigma(w) = n$ gibt, dann kann $w$ effektiv aus $n$ konstruiert werden.*

Es ist also bei Gödelisierungen keine Bijektion zwischen $\Sigma^*$ und $\mathbb{N}$ gefordert! Es gibt verschiedene Möglichkeiten spezielle Gödelisierungen anzugeben, von denen wir schon die Cantorsche Paarfunktion kennengelernt haben.

Die Paarfunktionen pair, paar und $\pi^{(k)}$ sind Gödelisierungen nur für Zahlentupel über $\mathbb{N}$ mit fester Dimension. Wenn wir allerdings endliche Zahlenfolgen oder Zeichenketten variabler Länge kodieren möchten, so taugen diese Abbildungen herzlich wenig.

Kurt Gödel verwendete als erster die Kodierung über die eindeutige Zahlenrepräsentation mit Hilfe von Primzahlen. Diese Kodierung $\nu_{\mathbb{N}-seq}$ ordnet zunächst jeder endlichen Zahlenfolge

$$\alpha_1, \alpha_2, \ldots, \alpha_m \text{ mit } \alpha_i \in \mathbb{N}$$

ihre sogenannte Gödelnummer

$$\nu_{\mathbb{N}-seq}(\alpha_1, \alpha_2, \ldots, \alpha_m) := \prod_{i=1}^{m} p_i^{\alpha_i + [i = m]} - 1$$

zu, wobei $p_i$ die $i$-te Primzahl der Folge $2, 3, 5, 7, 11, \ldots$ bezeichnet.

Da $\alpha_i = 0$ möglich sein soll, wird nur der Exponent $\alpha_m$ der letzten und höchsten Primzahl $p_m$ in dem Produkt um 1 erhöht! Hier macht sich die Iverson'sche Notation wieder nützlich. Damit alle Sequenzen über $\mathbb{N}$ als Zahlen aus $\mathbb{N}\setminus 0$ kodiert werden können, musste am Ende noch 1 abgezogen werden. Dadurch ist die leere Folge ohne ein einziges Element als 0 kodiert! Als Beispiel eine Liste der ersten Werte für kurze Folgen $u$ über $\mathbb{N}$:

| Folge $u$ | $\nu_\Sigma(u)$ |
|---:|---|
| 0 | 1 |
| 0,0 | 2 |
| 1 | 3 |
| 0,0,0 | 4 |
| 1,0 | 5 |
| 0,0,0,0 | 6 |
| 2 | 7 |
| 0,1 | 8 |
| 0,0,1 | 9 |
| ... | ... |

Einem Wort $w = x_{i_1} x_{i_2} \ldots x_{i_m}$ über dem Alphabet $\Sigma := \{x_0, x_1, \ldots, x_{n-1}\}$ wird dann die eindeutig bestimmte, natürliche Zahl $\nu_{prim}(w) = \nu_{\mathbb{N}-seq}(i_1, i_2, \ldots, i_m)$ zugeordnet. Dies führt zu einer ersten Gödelisierung von Zeichenketten:

**2.63 Definition**
*Sei $\Sigma := \{x_0, x_1, \ldots, x_{n-1}\}$ ein Alphabet, dann ist $\nu_{prim} : \Sigma^* \longrightarrow \mathbb{N}$ definiert durch:*

1. $\nu_{prim}(\lambda) := 0$,
2. $\nu_{prim}(w) := \prod_{j=1}^{m} p_j^{i_j + [j = m]} - 1$ *für Wörter der Form* $w = x_{i_1} x_{i_2} \cdots x_{i_m}$ *mit* $x_{i_j} \in \Sigma$.

Aus $n \in I\!N$ erhält man $\nu_{prim}^{-1}(n)$ durch fortgesetztes Dividieren.

Mit $\Sigma := \{x_0, x_1, x_2\}$ erhalten wir so $\nu_{prim}(x_2x_1x_0x_1) = 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^2 - 1 = 587$, $\nu_{prim}(x_1x_2x_0x_0) = 2^1 \cdot 3^2 \cdot 5^0 \cdot 7^1 - 1 = 125$, $\nu_{prim}(x_0) = 2^1 - 1 = 1$ und $\nu_{prim}(x_0x_0) = 2^0 \cdot 3^1 - 1 = 2$.

Da jede natürliche Zahl eine eindeutige Zerlegung als Produkt von Primzahlen besitzt, hat auch jedes $n \in I\!N$ ein Urbild unter der Umkehrfunktion $\nu_{prim}^{-1}$. Die Gödelisierung $\nu_{prim}$ ist also eine Bijektion zwischen $\Sigma^*$ und $I\!N$.

Als weitere Möglichkeit kann man aus der lexikalischen Ordnung der Zeichenketten ebenfalls eine eindeutige Gödelnummer $\nu_{lg-lex}(w)$ definieren.

**2.64 Definition**

*Für das geordnete Alphabet $(\Sigma, \prec), \Sigma := \{x_1, x_2, \ldots, x_n\}$ mit $x_1 \prec x_2 \prec \cdots \prec x_n$ sei $\nu_{lg-lex} : \Sigma^* \longrightarrow I\!N$ rekursiv definiert durch*

1. $\nu_{lg-lex}(wx_i) := n \cdot \nu_{lg-lex}(w) + i$, *falls* $w \in \Sigma^*, x_i \in \Sigma$,
2. $\nu_{lg-lex}(\lambda) := 0$.

Man zeigt leicht durch Induktion:

**2.65 Lemma**

*Für $\Sigma := \{x_1, x_2, \ldots, x_n\}$ und $x_{i_j} \in \Sigma$ gilt:*

$$\nu_{lg-lex}(x_{i_m}x_{i_{m-1}} \cdots x_{i_0}) = \sum_{j=0}^{m} i_j \cdot n^j.$$

**2.66 Beispiel**

Für $\Sigma := \{a, b, c\}$ mit $a \prec b \prec c$ ist mit dieser Kodierung dann $\nu_{lg-lex}(bca) = 28$.

Die Gödelisierung $\nu_{lg-lex}$ ist tatsächlich ebenfalls eine Bijektion zwischen $\Sigma^*$ und $I\!N$. Die Umkehrung kann am einfachsten durch einen Algorithmus angegeben werden, der für ein geordnetes Alphabet $(\Sigma, \prec)$ bei Eingabe einer beliebigen Zahl $n \in I\!N$ dasjenige Wort $w \in \Sigma^*$ erzeugt, welches in der Abzählung als $n$-tes vorkommt. Tatsächlich interessiert nur die Reihenfolge der Indizes der Symbole im Alphabet $\Sigma := \{x_1, x_2, \ldots, x_b\}$, die als b-adische Zahl (siehe Definition 2.69) aufgefaßt werden kann.

**Stellensysteme für die Zahlendarstellung** Man findet einen verwandten Algorithmus bei den Definitionen der Stellensysteme für natürliche Zahlen! Dieser ist in Algorithmus 2.68 dargestellt. Auf den zwei wichtigsten Notationen natürlicher Zahlen, den $b$-nären ($b$-*ary*) oder den $b$-adischen ($b$-*adic*) Darstellungen zur Basis (*base*, *radix*) $b$ beruhen viele Kodierungen. Sie sind auch wegen der technischen Gegebenheiten bei realen Computern und elektronischen Schaltungen von besonderer Bedeutung.

**2.67 Definition**

*Die **$b$-näre Darstellung** benutzt die Symbole der Ziffernmenge $B := \{0, 1, \ldots, b-1\}$. Ist $b = 2$, so spricht man von einer **binären** Zahlendarstellung. Eine Zahl $n \in I\!N$ wird zur Basis $b = |B|$ durch eine Folge $a_ka_{k-1} \cdots a_0$ von Symbolen $a_i \in \{0, 1, \ldots, b-1\}$ notiert, wenn $n = \sum_{i=0}^{k} a_i \cdot b^i$ gilt. Wir notieren dies wie üblich durch $[a_ka_{k-1} \cdots a_0]_b = n$.*

Wie man aus einer Binärzahl $w \in \{0,1\}^*$ die Zahl $[w|_b \in I\!N$ gewinnt ist nach der Definition natürlich einfach. Umgekehrt gewinnt man die Binärdarstellung einer Zahl $n \in I\!N$ durch fortgesetztes Dividieren durch 2 und Notieren der Reste, wie im Algorithmus 2.68 für die $b$-nären Darstellungen allgemein dargestellt.

**2.68 Algorithmus ($b$-näre Darstellung einer Zahl $n \in I\!N$)**

Sei $B := \{0, 1, \ldots, b-1\}$ ein Alphabet von $b$ Ziffern und $n \in I\!N$ die Eingabe für den Algorithmus.

**begin**
    **repeat** (* wiederhole alles was vor **until** steht, bis die Bedingung dahinter erfüllt ist *)
        $m := \lfloor \frac{n}{b} \rfloor$; $r := n - m \cdot b$;
        schreibe $r$ links neben eventuell schon Geschriebenes;
        $n := m$
    **until**
        $n := 0$
    **end** (* repeat *)
**end.** (* Algorithmus *)

**2.69 Definition**

*Die $b$-**adische Darstellung** benutzt die Symbole der Ziffernmenge $B := \{1, 2, \ldots, b\}$. Ist $b = 2$, so spricht man von einer **dyadischen** Zahlendarstellung. Eine natürliche Zahl $n \in I\!N, n \neq 0$ wird zur Basis $b = |B|$ durch eine Folge $a_k a_{k-1} \cdots a_0$ von Symbolen $a_i \in \{1, \ldots, b\}$ notiert, wenn $n = \sum_{i=0}^{k} a_i \cdot b^i$ gilt. Wir notieren dann $[a_k a_{k-1} \cdots a_0]_{b-adic} = n$.*

*Von einer **monadischen** Darstellung spricht man, wenn $B = \{1\}$ ist. Statt der Ziffer 1 wird oft jedes beliebige Symbol $z$, nicht notwendig eine Ziffer, benutzt. Mit $b = 1$ wird dann eine natürliche Zahl $n \in \mathbb{Z}^+$ durch die Zeichenkette $1^n = \underbrace{11 \cdots 1}_{n}$ dargestellt.*

| |
|---|
| Will man die Null anders als mit dem leeren Wort $\lambda$ darstellen, so vereinbart man in der Regel, dass $n \in I\!N$ durch $0^{n+1} = \underbrace{00 \cdots 0}_{n+1}$ dargestellt wird. Das verwandte Alphabet ist dann $B = \{0\}$.<br>Wir bezeichnen dies als **modifizierte unäre** Darstellung. |

Für das Alphabet $\Sigma := \{x_1, x_2, \ldots, x_b\}$ könnte man eine $b$-adische Darstellung benutzen, und einem Wort $w = x_{i_1} x_{i_2} \cdots x_{i_m}$ mit $x_{i_j} \in \Sigma$ die Zahl $\nu_{b-adic}(w) := \sum\limits_{k=0}^{m-1} (i_{m-k}) b^k$ zuordnen. Mit dieser Definition hätte man nun eine eindeutige Gödelnummer für jedes Wort über $\Sigma^*$ definiert, denn es gibt keine führenden Nullen in dieser Darstellung. Die b-adischen Darstellungen sind, anders als die b-nären, stets eindeutig! Sieht man etwas genauer hin, so stellt sich heraus, dass $\nu_{b-adic}(w) = \nu_{lg-lex}(w)$ ist, und erkennt die lexikalische Ordnung über $\Sigma^*$ als Anordnung hinsichtlich der (durch die Hintereinanderschreibung der Indizes der in $w \in \Sigma^*$ verwendeten Symbole aus $\Sigma$) $b$-adisch repräsentierten Zahlen!

Es ergibt sich so für die natürlichen Zahlen bis zur acht die folgende Liste binärer und dyadischer Zahlendarstellungen, wobei die Binärzahlen stets ohne führende Nullen geschrieben wurden:

| $b$-när | $n \in \mathbb{N}$ | $b$-adisch |
|---:|:---:|---:|
| 0 | 0 | undef. |
| 1 | 1 | 1 |
| 10 | 2 | 2 |
| 11 | 3 | 11 |
| 100 | 4 | 12 |
| 101 | 5 | 21 |
| 110 | 6 | 22 |
| 111 | 7 | 111 |
| 1000 | 8 | 112 |
| $\vdots$ | $\vdots$ | $\vdots$ |

Mit Algorithmus 2.70 lassen sich die $b$-adischen Darstellungen von natürlichen Zahlen $n \in \mathbb{N}$ gewinnen, und damit auch die Wörter $w \in \Sigma^*$, die in der lexikalischen Ordnung $\prec^{\mathrm{lg-lex}}$ zu einem geordneten Alphabet $(\{x_1, x_2, \ldots, x_b\}, \prec)$ an $n$-ter Stelle stehen. Dieses Verfahren ähnelt stark dem Algorithmus 2.68 zur Umwandlung einer Zahl in ihre $b$-näre Darstellung.

**2.70 Algorithmus (b-adische Darstellung einer natürlichen Zahl)**
Sei $B := \{1, 2, \ldots, b\}$ ein Alphabet von Ziffern und $n \in \mathbb{N}$ mit $n \neq 0$ die Eingabe für den Algorithmus.

```
begin
    repeat
        m := ⌊n/b⌋; r := n − m · b;
        if r ≠ 0
            then
                begin
                schreibe r links neben eventuell schon Geschriebenes
                n := m
                end
            else
                begin
                schreibe b links neben eventuell schon Geschriebenes;
                n := m − 1
                end
        end (* if *)
    until
        n := 0
    end (* repeat *)
end. (* Algorithmus *)
```

Würden wir das gleiche Prinzip für $w = x_{i_1} x_{i_2} \cdots x_{i_m}$ mit $x_{i_j} \in \Sigma := \{0, 1, \ldots, b-1\}$ die $b$-näre Darstellung für eine ähnlich definierte Funktion $g_{b-ary} : \Sigma^* \longrightarrow \mathbb{N}$ verwenden, so gäbe es zwar zu jeder Zahl $n \in \mathbb{N}$ ein Wort $w \in \Sigma^*$ mit $g_{b-ary}(w) = n$, aber wegen der führenden Nullen wäre dieses nicht eindeutig bestimmt. Die Abbildung $g_{b-ary} : \Sigma^* \longrightarrow \mathbb{N}$ wäre zwar total aber nicht injektiv und wäre somit keine Gödelisierung. Man müsste in diesem Falle dann noch die führenden Nullen ausschließen und könnte erst so eine Bijektion zwischen $B \setminus \{0\} \cdot B^*$ und $\mathbb{N}$ erklären. Hierbei ist es jedoch von Nachteil, dass nicht alle Wörter über dem Alphabet $\Sigma$ auch verwendet werden dürfen.

In der Regel werden wir bei unseren Anwendungen die genaue Definition einer Gödelisierung selten wirklich benötigen. Es reicht in den meisten Fällen zu wissen, dass eine solche existieren muss. Daher werden wir davon abstrahieren und die gewählte Gödelisierung einheitlich bezeichnen.

**2.71 Definition**

*Gödelisierungen werden von nun an immer als bijektive Abbildungen angesehen und – sofern das Alphabet $\Sigma$ unmissverständlich festgelegt ist – durch $\langle\ \rangle : \Sigma^* \longrightarrow \mathbb{N}$ notiert.*

**2.72 Beispiel**

Mit $\langle abbab \rangle$ wird diejenige natürliche Zahl dargestellt, die gemäß der beliebig gewählten, aber eindeutig festgesetzten Gödelisierung, die Gödelnummer von $abbab$ ist.

In den späteren Fällen wird $\mathbb{N} = \text{range}(\langle\ \rangle)$ oft sogar durch $\text{range}(\langle\ \rangle) := G^*$, für ein meist sehr kleines Alphabet $G$ (in der Regel $G = \{0, 1\}$) ersetzt werden. Die lexikalische Ordnung auf $G^*$ ist total, und definiert eine weitere Gödelisierung $\nu_{lg-lex} : G^* \longrightarrow \mathbb{N}$, die dann der Abbildung $\langle\ \rangle$ nachgeschaltet werden kann, damit letztlich $\nu_{lg-lex} \circ \langle\ \rangle$ die verlangte Gödelisierung eines Wortes $w$ in $\mathbb{N}$ darstellt.

Will man Wortmengen, d.h. formale Sprachen oder kurz Sprachen, definieren, so gelingt das durch Aufzählung aller Elemente nur bei endlichen Mengen. Für unendliche Mengen ist natürlich eine endliche Beschreibung für diese Mengen, d.h. aller zu ihr gehörenden Elemente, nötig. In späteren Kapiteln werden dazu noch Kalküle, Grammatiken und auch die verschiedensten Typen von Automaten erklärt werden.

Für die am meisten verwendeten und wichtigen Wortmengen, die regulären Mengen, wird dazu gerne der endliche Automat verwendet.

# 3 Endliche Automaten und reguläre Mengen

Gewöhnlich wird unter einem Automaten oder einer Maschine ein mechanisches oder elektronisches Gerät verstanden, bei dem ausschließlich die Funktion ausschlaggebend für seine Beurteilung ist. Mechanische Geräte wie Hammer, Lupe oder Schreibmaschine werden bei komplexeren Funktionen von schlichten Werkzeugen unterschieden. Ähnliches findet man bei elektronischen Geräten, wie zum Beispiel bei der Unterscheidung zwischen Transistor, Solarzelle oder Computer. Im täglichen Leben finden wir an fast jeder Straßenecke Automaten: Zigarettenautomaten, Fahrkartenautomaten, Musikbox, Glücksspielautomat; aber auch zu Hause: Telefon, CD-Spieler, Fernseher oder Anrufbeantworter. Gemeinsam sind all diesen realen Maschinen folgende Charakteristika: **Eingabeinformationen** (z.B.: Tastendruck, Münzeinwurf, Wählaktion, o.ä.), **interne Verarbeitung** ohne Rückgriff auf externe weitere Informationsquellen (z.B.: Prüfen der eingeworfenen Münzen, Berechnung des Fahrpreises auf Grund der Zielangabe, Summation der eingeworfenen Beträge) und **Ausgabeinformationen** (z.B.: Geldauswurf, Fahrkarte und Wechselgeld, Zigaretten, o.ä.). All diese Funktionen werden in der Regel unzweideutig stets in der gleichen Weise ausgeführt.

## 3.1 Deterministische endliche Automaten

Um nun von den nicht die Funktion betreffenden Aspekten wie Material, Farbe oder Herstellungspreis absehen zu können, wird in der Theoretischen Informatik von realen Bauweisen abstrahiert und nur Wesentliches beschrieben. Wir beginnen mit dem einfachsten Modell, dem **endlichen Automaten**. Dieser modelliert in der Regel eine Maschine, die ihre internen Zustände auf Grund von festgelegten Eingaben ändern kann und *keine Ausgabe* erzeugt. Deterministische endliche Automaten werden aber auch mit Ausgabefunktionen versehen und heißen dann nach ihren Erfindern Moore-Automat oder Mealy-Automat.

**3.1 Definition**

*Ein **deterministischer endlicher Automat** (**DFA** für **d**eterministic **f**inite **a**utomaton) wird durch ein 5-Tupel $A := (Z, \Sigma, \delta, z_0, Z_{end})$ beschrieben, wobei*

$Z$ *eine endliche Menge von **Zuständen** ist,*

$\Sigma$ *ein endliches Alphabet von **Eingabesymbolen** ist,*

$\delta :\subseteq Z \times \Sigma \longrightarrow Z$ *ist die nicht notwendigerweise totale **Überführungsfunktion**,*

$z_0 \in Z$ *der **Startzustand** ist und*

$Z_{end} \subseteq Z$ *die Menge der **Endzustände** ist.*

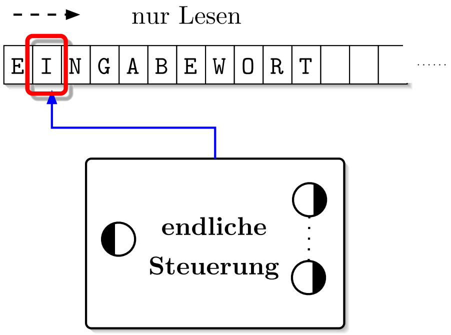



Abbildung 3.1: Grundschema des deterministischen endlichen Automaten

| |
|---|
| Zustände werden meist mit $z_0, z_1, \ldots$ oder mit $p_i, q_j, \ldots$, etc. bezeichnet. Für Eingabesymbole werden üblicherweise Buchstaben vom Anfang des lateinischen Alphabetes verwendet, also $a, b, c, \ldots$, o.ä. |

Wie schon im einleitenden Beispiel gezeigt, veranschaulichen wir einen DFA durch seinen **Zustandsgraphen**. Dies ist ein Kanten bewerteter gerichteter Graph, dessen Knoten eineindeutig den Zuständen des DFA zugeordnet sind.

Da die Knotenmenge des Zustandsgraphen isomorph zu der Zustandsmenge des Automaten ist, wollen wir hier jeden Knoten als einen Kreis um die Zustandsbezeichnung zeichnen: $z$. Startzustände werden hierbei als $z$ und Endzustände als $z$ gezeichnet. Eine mit dem Zeichen $x$ beschriftete Kante $z_1 \xrightarrow{x} z_2$ vom Knoten $z_1$ zum Knoten $z_2$ wird genau dann gezeichnet, wenn $\delta(z_1, x) = z_2$ gilt.

Einen DFA kann man sich als Maschine vorstellen, die ein Eingabeband besitzt auf dem in einzelne Felder jeweils Symbole $x_i \in \Sigma$ geschrieben sind, die in ihrer Folge von links nach rechte jenes Wort $w = x_1 x_2 \ldots x_n$ bilden, das als Eingabe des DFA dient. Um diese Eingabe verarbeiten zu können, besitzt der DFA einen Lesekopf, der auf dem Eingabeband jeweils ein Feld besuchen kann, dessen Beschriftung liest und in Abhängikeit von diesem Symbol und dem momentanen Zustand entsprechend der Übergangsfunktion $\delta$ in den dadurch bestimmten Folgezustand wechselt. Der DFA startet dazu in seinem Startzustand $z_0$, liest das Eingabewort von links nach rechte in der beschriebenen Weise und akzeptiert es, wenn er nach Lesen des letzten Symbols des Eingabewortes in einem Endzustand ist.

Diese informale Beschreibung wird nun mathematisch präziser, also formal, formuliert werden.

### 3.2 Definition

*Zu einem gegebenen DFA $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ definieren wir rekursiv die **erweiterte Überführungsfunktion** $\widehat{\delta} : Z \times \Sigma^* \longrightarrow Z$ durch:*

$$\widehat{\delta}(z, xw) := \widehat{\delta}(\delta(z, x), w)$$

*für alle Zustände $z \in Z$, alle Symbole $x \in \Sigma$ und alle Wörter $w \in \Sigma^*$, sowie*

$$\forall\, z_i \in Z : \widehat{\delta}(z_i, \lambda) := z_i.$$

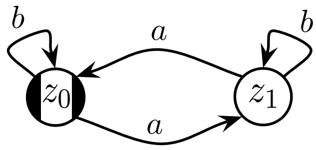



Abbildung 3.2: Beispiel-DFA

*Die von dem DFA $A$* ***akzeptierte Sprache*** *ist die Menge*

$$L(A) := \{w \in \Sigma^* | \widehat{\delta}(z_0, w) \in Z_{end}\}.$$

Offensichtlich kann ein DFA auch die leere Menge $\emptyset$ akzeptieren. Dazu muss nur die Menge $Z_{\text{end}}$ der Endzustände als leere Menge definiert werden, was ja nicht ausgeschlossen wurde!

Die Funktion $\widehat{\delta}$ ist tatsächlich eine Erweiterung der Funktion $\delta$, denn für alle Zustände $z \in Z$ und alle Symbole $x \in \Sigma$ gilt ja $\widehat{\delta}(z_i, x) = \delta(z, x)$. Ferner gilt offensichtlich

$$\widehat{\delta}(z, x_1 x_2 \cdots x_n) = \delta(\cdots \delta(\delta(z_i, x_1), x_2) \cdots, x_n).$$

Es ist üblich, anstelle der erweiterten Funtion $\widehat{\delta}$ etwas unpräzise nur $\delta$ zu notieren, sofern dadurch keine Verwirrung entsteht. Auch wir wollen uns diese Notationkünftig verwenden.

**3.3 Beispiel**

Betrachten wir den Zustandsgraphen aus Abbildung 3.2. Es stellt einen DFA dar, der die Menge aller Wörter $w \in \{a, b\}^*$ erkennt, die eine gerade Anzahl von Symbolen $a$ enthalten. Auch das leere Wort $\lambda$ ist ein solches Wort und wird von dem Automaten akzeptiert, ohne den Startzustand zu verlassen. Dies ist möglich, da der Startzustand gleichzeitig ein Endzustand ist.

> Bedenken Sie bei der Betrachtung eines endlichen Automaten immer genau, ob das Akzeptieren des leeren Wortes $\lambda$ gewünscht ist oder nicht!

Neben der allgemeinsten Form des deterministischen endlichen Automaten (DFA) nach Def. 3.1 gibt es spezielle Modelle, die besonderen Einschränkungen unterworfen sind. Zum Beispiel war in der Definition des DFA nicht gefordert worden, dass es mindestens einen Endzustand geben muss, oder es zu jedem Eingabesymbol in jedem Zustand auch ein definierter Übergang existiert: Weder die Überführungsfunktion $\delta$, noch folglich deren Erweiterung $\widehat{\delta}$, musste eine totale Funktion sein!

**3.4 Definition**

*Ein DFA $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ heißt:*

1. ***vollständig*** *(Abk.: vDFA) genau dann, wenn für jedes $(p, x) \in Z \times \Sigma$ ein $q \in Z$ existiert, so dass $q = \delta(p, x)$ ist.*

2. ***initial zusammenhängend*** *(Abk.: izDFA) genau dann, wenn zu jedem Zustand $p \in Z$ ein Wort $w \in \Sigma^*$ existiert, mit $p = \widehat{\delta}(z_0, w)$.*

*Zwei verschiedene DFA, $A$ und $B$, heißen genau dann* ***äquivalent****, wenn sie die gleiche Sprache akzeptieren, d.h. $L(A) = L(B)$ ist.*

In vollständigen DFA's ist die Übergangsfunktion tatsächlich total, also eine Abbildung. Die Frage, ob es zu jedem DFA einen äquivalenten vollständigen DFA oder einen äquivalenten initial zusammenhängenden DFA gibt, werden wir in Kürze beantworten.

Nachdem wir gemäß Definition 2.45 jede Teilmenge von $\Sigma^*$ als *formale Sprache* bezeichnen, jede Menge von Mengen in der Regel als *Klasse von Mengen*, wollen wir nun unter den Klassen von formalen Sprachen sogenannte „Familien von Sprachen“, kurz Sprachfamilien, kennzeichnen:

**3.5 Definition**

*Eine Klasse $\mathcal{L}$ von formalen Sprachen wird* ***Sprachfamilie*** *genannt, wenn folgende Bedingungen erfüllt sind:*

1. $\mathcal{L} \neq \emptyset$.
2. *Es existiert ein abzählbar unendliches Alphabet $\Gamma$, so dass für jede Sprache $L \in \mathcal{L}$ ein endliches Alphabet $\Sigma \subseteq \Gamma$ existiert mit $L \subseteq \Sigma^*$.*
3. $\exists L \in \mathcal{L} : L \neq \emptyset$.

**3.6 Beispiel**

Offenbar ist die Klasse *aller* Wortmengen **keine** Sprachfamilie, da es für diese unendlich vielen Sprachen kein gemeinsames endliches Alphabet $\Sigma$ gibt. Zum Beispiel gibt es für die unendliche, abzählbare Menge $\Gamma$ kein endliches Alphabet $\Sigma$ mit $\Gamma \subseteq \Sigma$. Die Klasse aller endlichen Mengen wie auch die Klasse $2^{\Sigma^*}$ aller Sprachen über dem Alphabet $\Sigma$ sind offensichtlich Sprachfamilien, jedoch ohne sonderlich nutzbare Struktur. Alle PROLOG-Programme bilden eine Sprachfamilie, wenn jedes einzelne PROLOG-Programm als eine Zeichenkette über dem ASCII-Zeichensatz aufgefasst wird (auch die Wortzwischenräume „*blanks*“ sind einzelne Zeichen!).

**3.7 Definition**

*Die Mengen von Wörtern, die von einem DFA akzeptiert (erkannt) werden können, nennt man* ***reguläre Mengen*** *und bezeichnet die gesamte Familie derselben mit $\mathcal{Reg}$. Mit $\mathcal{Akz}(\Sigma)$ wird die Familie aller jener Sprachen $L \subseteq \Sigma^*$ bezeichnet, die von deterministischen endlichen Automaten mit dem Eingabe-Alphabet $\Sigma$ akzeptiert werden können. Mithin gilt:*

$$\mathcal{Akz}(\Sigma) = \{L \subseteq \Sigma^* \mid L = L(A) \text{ für einen DFA } A\}$$

*und*

$$\mathcal{Reg} = \bigcup_{\substack{\Sigma \text{ ist endl.} \\ \text{Alphabet}}} \mathcal{Akz}(\Sigma).$$

Nach Definition 3.4 können wir auch spezielle Typen von deterministischen endlichen Automaten verwenden, wissen bisher aber noch nicht, ob die Beschränkung auf vDFA oder izDFA vielleicht das Akzeptieren einiger regulärer Mengen verhindert. Dass dem zum Glück nicht so ist, werden wir im Verlauf der nächsten Untersuchungen noch feststellen.

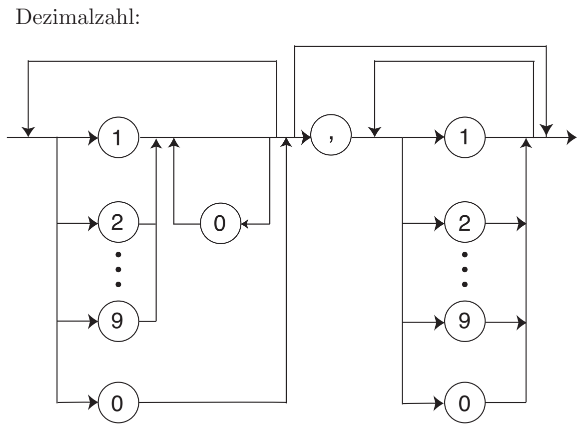



Abbildung 3.3: Syntaxdiagramme

## 3.2 Nichtdeterministische endliche Automaten

Syntaxdiagramme zu Programmiersprachen wie zum Beispiel PASCAL oder MODULA geben an, auf welche Weise ein korrekter Ausdruck entsprechend der Definition geschrieben werden kann. Dies Vorgehen ist wahlfrei oder nichtdeterministisch in dem Sinne, dass alles erlaubt ist, was nicht verboten wurde.

**3.8 Beispiel**
In Abbildung 3.3 wird die Syntax für Dezimalzahlen durch ein rekursionsfreies Syntaxdiagramm angegeben.

Die Interpretation dieses Diagramms geschieht auf folgende Weise: Links beginnend dürfen die Kanten, den Pfeilen folgend, gegangen werden, wobei die in den Kreisen (oder Ovalen) stehenden Symbole aufgeschrieben werden sollen, und am Ende das Diagramm rechts verlassen werden muss. An den Abzweigungen kann jeder beliebige Weg gewählt werden, und jede so notierbare Zeichenkette ist eine in der betreffenden Programmiersprache erlaubte Dezimalzahl, wobei lediglich eine eventuelle Obergrenze in der Zahl der Ziffern mit dem Syntaxdiagramm nicht formuliert wurde. Aufmerksame Betrachter(innen) finden zum Beispiel für die Ziffernfolge 2000 mehr als nur einen Weg durch obiges Diagramm: Die kleine Rückwärtsschleife mit der Ziffer „0“ kann nach Besuchen des Knotens 2 dreimal oder nur zweimal durchlaufen werden. In letzterem Fall muss die dritte Null der Ziffernfolge 2000 – nach Benutzen des oberen Rückwegs nach ganz links – dann über den unteren Weg gewählt werden. Ganz diesem Beispiel entsprechend, kann man sich einen endlichen nichtdeterministischen Automaten als einen bewerteten, gerichteten Graphen vorstellen. Dieser besitzt extra ausgezeichnete Start- und Zielknoten, und beschreibt dann die Menge aller jener Zeichenketten, die durch Aneinanderketten aller Kantenbewertungen entstehen können, die entlang gerichteter Kanten auf einem Weg von einem beliebigen Startknoten zu einem ebenso beliebigen Zielknoten zu finden sind. So wie ein Taxifahrer in einer Großstadt im allgemeinen mehrere Möglichkeiten hat, seinen Kunden zum Ziel zu fahren, gibt es in solch einem gerichteten gerichteter Graphen ebenfalls mehrere, oft beliebig viele, Möglichkeiten von einem Start- zu einem Zielknoten zu gelangen. In vielen Fällen vereinfachen sich die Definitionen von formalen Sprachen auf diese Weise erheblich, auch wenn damit die anschaulichere Vorstellung vom endlichen Automaten als einer real existierenden Maschine, aufgegeben werden muss.

Eine andere Vorstellung sieht einen nichtdeterministischen endlichen Automaten lediglich als eine Erweiterung des bekannten DFA. In dieser Erweiterung gibt die Überführungsfunktion bei Eingabe eines Symbols in einem Zustand möglicherweise mehrere Folgezustände an. Beide Vorstellungen haben ihre Vorteile und wir geben beide Definitionsvarianten zum Vergleich an, werden danach aber stets die „graphenorientierte“ Definition 3.10 bevorzugen.

**3.9 Definition**

*Ein **nichtdeterministischer endlicher Automat** (**NFA**, engl. **n**ondeterministic **f**inite **a**utomaton) wird spezifiziert werden durch ein Tupel $A := (Z, \Sigma, \delta, Z_{\text{start}}, Z_{\text{end}})$, wobei*

> $Z$ *eine endliche Menge von **Zuständen** ist,*
>
> $\Sigma$ *ein endliches Alphabet von **Eingabesymbolen** ist,*
>
> $\delta : Z \times \Sigma \longrightarrow 2^Z$ *eine Abbildung ist, die **Überführungsfunktion** genannt wird,*
>
> $Z_{\text{start}} \subseteq Z$ *die **Menge der Startzustände** ist und*
>
> $Z_{\text{end}} \subseteq Z$ *die **Menge der Endzustände** ist,*

*Die Überführungsfunktion $\delta$ ist deshalb eine totale Funktion, weil dort wo die Über-führungsfunktion eines DFA undefiniert wäre, hier als Bild die leere Mange $\emptyset$ gewählt werden kann. Wie beim DFA auch, wird die Abbildung $\delta$ verallgemeinert, hier aber etwas anders, weil wir es hier mit Zustandsmengen zu tun haben.*

$\widehat{\delta} : 2^Z \times \Sigma^* \longrightarrow 2^Z$ *sei erklärt durch:*

$$\widehat{\delta}(Z', xw) := \bigcup_{z \in Z'} \widehat{\delta}(\delta(z, x), w)$$

*für alle $w \in \Sigma^*$, alle $x \in \Sigma$ und jede Teilmenge $Z' \subseteq Z$, sowie*

$$\forall\, Z' \subseteq Z : \widehat{\delta}(Z', \lambda) := Z'.$$

*$L(A) := \{w \in \Sigma^* \mid \widehat{\delta}(Z_{\text{start}}, w) \cap Z_{\text{end}} \neq \emptyset\}$ ist die von $A$ akzeptierte Sprache.*

Wenn wir diese Definition benutzen, so müssen wir einen NFA, der genau das Wort $w \in \Sigma^*$ akzeptieren soll, mit $|w| + 1$ Zuständen modellieren, auch wenn diesem Wort in dem Zustandsgraphen genau ein Pfad zugeordnet ist. Auch darf das leere Wort $\lambda$ nicht als Kanteninschrift verwendet werden. All dies schränkt einfache Darstellungsmöglichkeiten ein. Wie wir sehen werden, ist dies nicht nötig, und die folgende Definition des nichtdeterministischen endlichen Automaten ist die allgemeinste in dieser Form. Sie orientiert sich mehr an dem Zustandsgraphen als an der funktionalen Sichtweise mit einer Überführungsfunktion.

**3.10 Definition**

*Ein **nichtdeterministischer, endlicher Automat** (**NFA**, engl. **n**ondeterministic **f**inite **a**utomaton) ist ein Tupel $A := (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$, wobei die einzelnen Größen wie folgt erklärt sind:*

> $Z$ *ist eine endliche Menge von Zuständen,*

*$Z_{\text{start}} \subseteq Z$ ist die Menge der Startzustände,*

*$Z_{\text{end}} \subseteq Z$ ist die Menge der Endzustände,*

*$\Sigma$ ist ein endliches Alphabet von Eingabesymbolen und*

*$K \subseteq Z \times \Sigma^* \times Z$ ist die endliche Relation (vergl. Def. 2.10) der Zustandsübergänge. Ein Element $(p, \lambda, q) \in K$ wird als **$\lambda$-Kante** bezeichnet. Ein NFA ohne $\lambda$-Kanten wird als $\lambda$-freier NFA bezeichnet.*

Um mit diesen Automaten arbeiten zu können, benötigen wir noch die Begriffe der *Rechnung* und der *Erfolgsrechnung* sowie den daraus resultierenden Begriff der *akzeptierten Sprache* eines NFA.

**3.11 Definition**

*Eine Kantenfolge $p := (z_0, x_1, z_1), (z_1, x_2, z_2), \ldots, (z_{n-1}, x_n, z_n)$, mit $(z_{i-1}, x_i, z_i) \in K$, für $0 < i \leq n$, heißt **Rechnung** von $A$ für das Wort $|p| := w := x_1 x_2 \ldots x_n \in \Sigma^*$. Dies wird kurz notiert durch: $z_0 \xrightarrow[w]{*} z_n$ und für jedes $w \in \Sigma^*$ ist $\xrightarrow[w]{*}$ eine Teilmenge von $Z \times Z$ und kann somit als eine Relation auf $Z$ angesehen werden.*

*Eine **Erfolgsrechnung** für das Wort $w$, ist eine Rechnung, bei der $z_0 \in Z_{\text{start}}$ und $z_n \in Z_{\text{end}}$ ist, oder $w = \lambda$ und $z_0 = z_n$ mit $z_n \in Z_{\text{start}} \cap Z_{\text{end}}$.*

*$L(A) := \{w \in \Sigma^* \mid \exists z_0 \in Z_{\text{start}} : \exists z_n \in Z_{\text{end}} : z_0 \xrightarrow[w]{*} z_n\}$ bezeichnet die von $A$ **akzeptierte Sprache**.*

Es werden also genau solche Wörter $w \in \Sigma^*$ akzeptiert, für die in dem Automaten $A$ eine Erfolgsrechnung existiert. Das leere Wort wird immer akzeptiert, wenn – wie beim DFA – Anfangs- und Endzustand identisch sind; oder ein ausschließlich $\lambda$-Kanten enthaltender Pfad von einem Startzustand zu einem Endzustand im Zustandsgraphen des NFA existiert. Es ist also möglich, dass auch ein $\lambda$-freier NFA das leere Wort akzeptiert!

Einen speziellen nichtdeterministischen endlichen Automaten $A$ kann man natürlich immer durch explizite Angabe der fünf Mengen $Z, \Sigma, K, Z_{\text{start}}$ und $Z_{\text{end}}$ notieren, besser ist aber meist auch hier die Angabe des Zustandsgraphen, der ähnlich erklärt wird wie beim DFA. Bei kleinen Automaten ist dies in der Regel auch übersichtlicher. Automaten, die einer bestimmten Einschränkung genügen sind in manchen Fällen einfacher zu benutzen, besonders bei Beweisen oder beim Umsetzen in Algorithmen. Daher geben wir hier zwei besondere Klassen von nichtdeterministischen endlichen Automaten an.

**3.12 Definition**

*Ein NFA $A := (Z, K, Z_{\text{start}}, Z_{\text{end}})$ heißt*

1. ***buchstabierend** genau dann, wenn $K \subseteq Z \times \Sigma \times Z$ ist,*
2. ***$\lambda$-frei** genau dann, wenn $K \subseteq Z \times \Sigma^+ \times Z$ ist.*

*Zwei NFA, $A$ und $B$, heißen genau dann **äquivalent**, wenn sie die gleiche Sprache akzeptieren, d.h. $L(A) = L(B)$ ist.*

Da wir die Zustandsgraphen von endlichen Automaten nicht nur als einfache oder bequeme Darstellungen ansehen wollen, sondern bei deren Benutzung auch Analysen vornehmen wollen, werden wir die hier benötigten mathematischen Begriffe einführen und erläutern. Wir schließen dabei an die mathematischen Grundbegriffe aus Kapitel 2 an.

Relationen bilden ein wichtiges und anschauliches Hilfsmittel, nicht nur bei der Darstellung von endlichen Automaten, sondern auch bei der Beschreibung und Formulierung von anderen Modellen oder Algorithmen. Jede binäre Relation lässt sich als gerichteter Graph auffassen. Algorithmen für Graphen können umgekehrt auch oft bei Fragen zu Relationen sinnvoll genutzt werden.

**3.13 Definition**

*Sei $R$ eine binäre Relation auf $A \times B$. Der* ***Graph*** *$G(R)$ der Relation ist der* ***gerichtete Graph*** *(directed graph) $G(R) := (V, E)$ mit der Menge der* ***Knoten*** *(Ecken, vertices) $V := A \cup B$ und der Menge $E := R \subseteq V \times V$ der* ***gerichteten Kanten*** *(edges). Die Kante $(a, b) \in E$, notiert als $a \longrightarrow b$, wird also genau dann gezeichnet, wenn $(a, b) \in R$ gilt. Notiert wird ein gerichteter Graph $G = (V, E)$ oft durch seine* ***Adjazenzmatrix*** *$C_G \in \{0, 1\}^{A \times B}$. Dies ist eine Matrix, deren Zeilen, mit den Elementen des Vorbereichs von $R$, also $A$, und deren Spalten, mit den Elementen des Nachbereichs von $R$, also $B$, bezeichnet werden. Die einzelnen Matrizenelemente $C_G(x, y)$ sind definiert durch:*

$$\forall x \in A : \forall y \in B : C_G(x, y) := [(x, y) \in E].$$

Mit dieser Notation, kann die Komposition von Relationen (vergl. Def. 2.10) durch ein modifiziertes Produkt der Adjazenzmatrizen gebildet werden. Wir benutzen das gewöhnliche Matrizenprodukt, nur interpretieren wir die verwendeten Operationen Summe und Multiplikation anders!

**3.14 Definition**

*Seien $A \in \{0, 1\}^{m \times n}$ und $M_2 \in \{0, 1\}^{n \times r}$ spezielle Matrizen über $\mathbb{Z}$, die wir in diesem Zusammenhang als* ***boolesche Matrizen*** *bezeichnen wollen, da die Werte 0 und 1 in ihren Komponenten wie boolesche Wahrheitswerte benutzt werden sollen. Das* ***boolesche Matrizenprodukt*** *von $A$ und $B$ ist für jedes Element $C(i, j)$, $1 \leq i \leq m$, $1 \leq j \leq r$ für $C := A \otimes B$ definiert durch:*

$$C(i, j) := \max\left\{\min\left\{A(i, k), B(j, k)\right\} \middle| 1 \leq k \leq n\right\}.$$

Statt $x + y$ in $I\!R$ bilden wir also $\max\{x, y\}$ und anstelle von $x \cdot y$ in $I\!R$ bilden wir $min\{x, y\}$. Interpretiert man die Zahlen 0 und 1 als Wahrheitswerte (1 für *wahr* und 0 für *falsch*), so entspricht die modifizierte Summe (max) über den Werten 0 und 1 nun gerade dem logischen Junktor $\vee$ (*„oder“/*) und das Produkt (min) dem logischen Junktor $\wedge$ (*„und“/*). Daher heißt diese Multiplikation eben auch boolesches Matrizenprodukt. Es passt zur Definition der Adjazenzmatrix $C_{G(R)}$ zum Graph der Relation $R$, mit

$$C_{G(R)}(x, y) = [(x, y) \in R] = \begin{cases} 1, & \text{falls } (x, y) \in R \\ 0, & \text{falls } (x, y) \notin R. \end{cases}$$

Mit dieser Darstellung erhalten wir für die Komposition $R_1 \cdot R_2$ der Relationen $R_1 \subseteq A \times B$ und $R_2 \subseteq B \times C$ dann $C_{G(R_1)} \otimes C_{G(R_2)} = C_{G(R_1 \cdot R_2)}$.

Ein kleines Beispiel soll dies veranschaulichen:

**3.15 Beispiel**

Seien $A := \{a, b\}$, $B := \{x, y, z\}$, $C := \{p, q\}$ und $R_1 \subseteq A \times B$, $R_2 \subseteq B \times C$ gegeben durch $R_1 := \{(a, x), (a, y), (b, y), (b, z), \}$ und $R_2 := \{(x, q), (y, p), (y, q), (z, q), \}$. Die Graphen der Relationen sind in Abbildung 3.4 dargestellt.

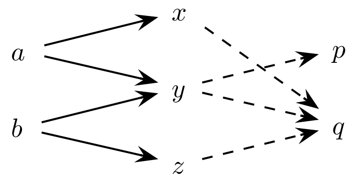



Abbildung 3.4: Die Graphen von $R_1$ (durchgehend) und $R_2$ (gestrichelt) aus Beispiel 3.15

Mit den Adjazenzmatrizen $C_{G(R_1)} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ und $C_{G(R_2)} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$ errechnet man leicht, dass deren boolesches Produkt gerade die Adjazenzmatrix $C_{G(R_1 \cdot R_2)} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ zum Graphen der Relationskomposition $R_1 \cdot R_2$ ergibt!

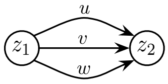
Wichtig bei dieser Auffassung ist, dass der Graph einer Relation keine doppelten Kanten zwischen den gleichen Punkten besitzt, denn eine Relation ist eine Menge von Paaren, und kann diese nicht mehrfach enthalten. Das wird erst möglich, wenn wir die einzelnen Kanten von einander unterscheiden können, was durch deren **Beschriftung** oder **Bewertung** geschieht. In nachfolgender Definition 3.16 wird ausdrücklich eine beliebige Menge $X$ von möglichen Bewertungen oder Kantenbeschriftungen gestattet, damit wir diese Kanten bewerteten Graphen möglichst flexibel handhaben können.

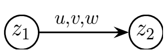
**3.16 Definition**
*Sei $K \subseteq A \times X \times B$ eine Relation „von $A$ nach $B$“ mit Kantenbewertungen aus der Menge $\Sigma$. Der **kantenbewertete gerichtete Graph** zu $K$ ist spezifiziert durch das Tripel $G_K := (V, X, K)$ mit Knotenmenge $V := A \cup B$. Eine bewertete Kante $(a, x, b) \in K$ von $G_K$ wird als $a \xrightarrow{x} b$ gezeichnet.*

Wie beim DFA ist auch die Knotenmenge des Zustandsgraphen eines NFA isomorph zu seiner Zustandsmenge und jeder Knoten wird wieder als Kreis um die Zustandsbezeichnung gezeichnet. Start- und Endzustände werden in der gleichen Weise dargestellt wie beim Zustandsgraphen eines DFA's. Eine formale Definition können wir nun mit Hilfe des Graphen zu der Relation $K$ eines NFA geben:

**3.17 Definition**
*Sei $A := (Z, \Sigma, K, Z_{start}, Z_{end})$ ein NFA, dann ist der Zustandsgraph zu $A$ der Graph $G_K := (Z, \Sigma^*, K)$.*

Da von einem Zustand $p$ in einem nichtdeterministischen endlichen Automaten nun mehrere verschiedene Kanten mit unterschiedlichen Wörtern als Beschriftung zum selben Zustand $q$ führen können, wird eine vereinfachte Darstellung bevorzugt:

Sind zum Beispiel die Kanten $(z_1, u, z_2), (z_1, v, z_2)$ und $(z_1, w, z_2)$ in $K$ enthalten, so notieren wir statt 

abkürzend 

.

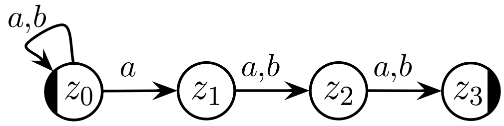



Abbildung 3.5: NFA, der alle Wörter akzeptiert, die an drittletzter Position ein $a$ haben

**3.18 Beispiel**

In Abbildung 3.5 ist der Zustandsgraph eines NFA, der genau diejenigen Wörter $w \in \{a,b\}^*$ akzeptiert, die an drittletzter Position das Symbol $a$ enthalten. Dieser endliche Automat akzeptiert also z.B. das Wort $abaaba$, *nicht* aber das Wort $abaabaa$.

Da jeder DFA ein spezieller NFA ist, kann natürlich jede reguläre Menge von einem NFA akzeptiert werden. Zu fragen ist jedoch, ob es Sprachen gibt, die ein NFA akzeptieren kann, die nicht regulär sind und wie überhaupt gezeigt werden kann, ob es überhaupt eine formale Sprache gibt, die nicht regulär ist und wie man dieses eventuell beweisen könnte! Wir werden dies Antwort hier nicht schuldig bleiben, und sogar mehrere, jedoch sicher nicht alle, Möglichkeiten zu deren Beantwortung darstellen.

## 3.3 Endliche Automaten mit Ausgabe

Neben den im vorigen Abschnitt definierten endlichen Automaten, die zum *Akzeptieren* von Wörtern bzw. Sprachen verwendet werden, werden wir hier solche definieren, die ihre Eingaben in passende Ausgaben *transformieren*. Wir unterscheiden dabei *Mealy-* und *Moore-Automaten* hinsichtlich der Weise, wie die Ausgabe bestimmt wird. Beide Modelle sind deterministische endliche Automaten mit Ausgabe (*finite state machines*), die *ohne* spezifizierte Endzustandsmenge auskommen, denn *jede* Eingabe über dem Eingabealphabet soll eine eindeutig spezifizierte Ausgabe bewirken. Die Bezeichnung „Maschine“ (*machine*) wird anstelle von „Automat“ in der Regel immer dann benutzt, wenn es sich um *deterministische* Automaten handelt!

Beim Moore-Automaten wird ein einzelnes Symbol als Ausgabe jedesmal dann ausgegeben, wenn ein Zustand erreicht (oder durchlaufen) wird; die Ausgabe richtet sich dabei nur nach dem jeweiligen Zustand. Typische reale Automaten von diesem Typ sind die bekannten Fahrkartenautomaten. Dort kommt es nur darauf an welche Taste man gedrückt hat und wieviel Geld eingeworfen wurde, nicht aber, in welcher Reihenfolge die Münzen für den zu zahlende Betrag eingegeben wurden.

Beim Mealy-Automaten wird ein einzelnes Symbol als Ausgabe jeweils bei einer Transition, d.h. einem Zustandsübergang, getätigt. Ein Beispiel dafür war der Kugelautomat aus Kapitel 1. Auch die sogenannte gsm (*generalized sequential machine*) und der allgemeine Transduktor mit oder ohne akzeptierende Endzustände (*a-transducer*, *transducer*) arbeiten nach diesem Prinzip, können aber in jedem Schritt beliebige Zeichenketten ausgeben. Der folgende Transduktor ist tatsächlich eine gsm aber kein Mealy-Automat.

**3.19 Beispiel**

Der Transduktor aus Abbildung 3.6 transformiert jedes Eingabewort der Form $w_1 c w_2 c \ldots c w_n$, mit $w_i \in \{0,1\}^*$, in das entsprechende Ausgabewort der Form $\varphi(w_1) c w_2 c \varphi(w_3) c w_4 \ldots \varphi(w_{n-1}) c w_n$ , falls $n$ gerade

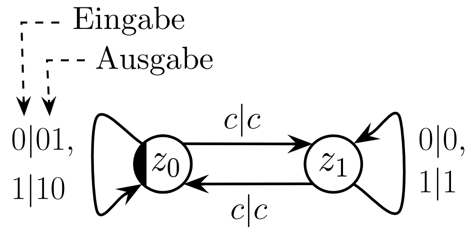



Abbildung 3.6: Ein Transduktor

ist, bzw. $\varphi(w_1)cw_2c\varphi(w_3)cw_4\ldots w_{n-1}c\varphi(w_n)$, falls $n$ ungerade ist. Hierbei wird $\varphi : \{0,1\}^* \to \{0,1\}^*$ als Homomorphismus mit $\varphi(0) = 01, \varphi(1) = 10$ benutzt.

**3.20 Definition**

1. *Ein* ***Moore-Automat*** *wird beschrieben durch $M = (Z, \Sigma, \Gamma, \delta, \rho, q_0)$, wobei $A := (Z, \Sigma, \delta, z_0, Z)$ ein vollständiger DFA, $\Gamma$ das* ***Ausgabealphabet*** *und $\rho : Z \longrightarrow \Gamma$ die* ***Ausgabefunktion*** *ist.*
2. *Die* ***Ausgabe*** *von $M$ für das Eingabewort $w := x_1x_2\ldots x_n$, mit $x_i \in \Sigma$, ist definiert durch das Wort $\rho(q_0)\rho(q_1)\ldots\rho(q_n)$ wobei $q_0, q_1, \ldots, q_n$ diejenige Folge von Zuständen ist, für die $\delta(q_i, x_{i+1}) = q_{i+1}, 0 \le i \le n$ gilt, also $(q_0, x_1, q_1), (q_1, x_2, q_2), \ldots, (q_{n-1}, x_n, q_n)$ eine Rechnung von $A$ für die Eingabe $x_1x_2\ldots x_n$ ist.*

| |
|---|
| Eine spezielle Endzustandsmenge gibt es bei Moore-Automaten nicht: jeder Zustand darf als Endzustand betrachtet werden, und da der Moore-Automat vollständig ist, kann jedem Wort aus $\Sigma^*$ eine Ausgabe zugeordnet werden! |

**3.21 Beispiel**

Wenn wir zwei Binärzahlen addieren wollen so schreiben wir diese gewöhnlich untereinander und beginnen *von rechts* die einzelnen Stellen zu addieren. Dabei ist die Summe von $1$ und $1$ eben die Binärzahl $10$ und man notiert $0$ mit $1$ als Übertrag. Die Summe $\binom{11101}{1011}$ liefert dann das Ergebnis $101000$.

Wollen wir dieses mit einem Moore-Automaten bewerkstelligen, so müsssen wir jeweils zwei übereinanderstehende Eingabeziffern mit den Symbolen $\binom{0}{0}$, $\binom{1}{0}$, $\binom{0}{1}$ und $\binom{1}{1}$ als eine Folge von kombinierten Eingabesymbolen kodieren und diese von rechts nach links einzeln der Maschine zuführen.

Die Summe $\binom{11101}{1011}$ wird also kodiert als $\binom{1}{0}\binom{1}{1}\binom{1}{0}\binom{0}{1}\binom{1}{1}$ und entspricht dann der umgekehrten Eingabesequenz: $\binom{1}{1}\binom{0}{1}\binom{1}{0}\binom{1}{1}\binom{1}{0}\binom{0}{0}$, die für einen eventuellen Übertrag am rechten Ende um das Symbol $\binom{0}{0}$ verlängert wurde. Die Ausgabe an den Zuständen muss der letzten Stelle der jeweils errechneten einfachen Summe entsprechen, während mit dem Folgezustand der zu addierende Übertrag gespeichert werden muss.

Abbildung 3.7 zeigt einen Moore-Automaten mit dem Eingabealphabet $\{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}$. Die Eingabe beginnt mit dem letzten Bitpaar der Addition und endet mit Eingabe eines zusätzlichen Paares $\binom{0}{0}$ für den letzten Übertrag.

Die Ausgabe ist bei $z_i$ gerade $i$ **mod** $2$, also bei den beiden Zuständen $z_0$ und $z_2$ jeweils $0$ und bei den beiden Zuständen $z_1$ und $z_3$ jeweils eine $1$.

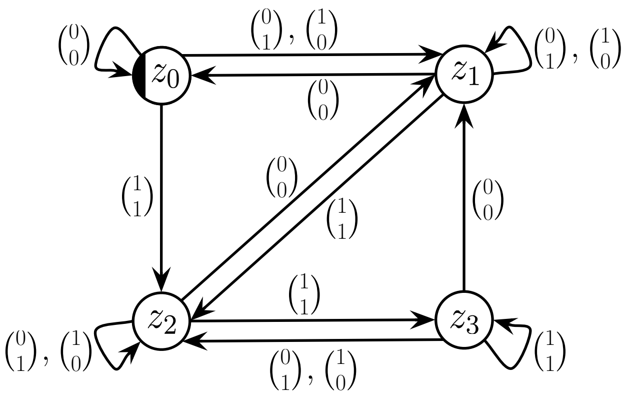


Abbildung 3.7: Ein Moore-Automat als Serienaddierer

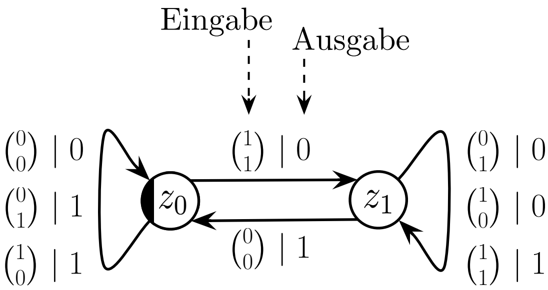



Abbildung 3.8: Ein Mealy-Automat als Serienaddierer

**3.22 Definition**

1. *Ein **Mealy-Automat** wird beschrieben durch $M = (Z, \Sigma, \Gamma, \delta, \rho, q_0)$, wobei $A := (Z, \Sigma, \delta, q_0, Z)$ ein vollständiger DFA, $\Gamma$ das **Ausgabealphabet** und $\rho : Z \times \Sigma \longrightarrow \Gamma$ die **Ausgabefunktion** ist.*

2. *Die **Ausgabe** von $M$ für das Eingabewort $w := x_1 \ldots x_n$, mit $x_i \in X$, ist definiert durch das Wort $\rho(q_0, x_1)\rho(q_1, x_2) \ldots \rho(q_{n-1}, x_n)$ wobei $q_0, q_1, \ldots, q_{n-1}$ eine Folge von Zuständen ist, für die $\delta(q_i, x_{i+1}) = q_{i+1}, 0 \leq i < n-1$ gilt, also $(q_0, x_1, q_1), (q_1, x_2, q_2), \ldots, (q_{n-1}, x_n, q_n)$ eine Rechnung von $A$ für die Eingabe $x_1x_2 \ldots x_n$ ist.*

**3.23 Beispiel (Mealy-Automat als Serienaddierer)**

Abbildung 3.8 zeigt einen Mealy-Automaten, also einen DFA mit Ausgabe, der einen Serienaddierer modelliert.

Mealy- und Moore-Automaten sind deterministische Maschinen, für die kein Endzustand spezifiziert wurde oder nötig gewesen wäre. Zu jedem Wort $w \in \Sigma^*$ gibt es eine eindeutige Übersetzung und diese wird durch den deterministischen Automaten funktional definiert. Mealy-Automaten werden oft bezüglich ihrer Ausgabefunktion zu $\rho : Z \times X \to Y^*$ erweitert. Solche Maschinen werden *gsm* (*generalized sequential machine*) genannt. Man definiert sogar die Erweiterung zu NFA's mit Ausgaben vom

Mealy-Typ, den sogenannten akzeptierenden Transduktoren, deren Studium wir hier nicht mehr weiter verfolgen wollen.

Das folgende Ergebnis über akzeptierende endliche Automaten ist sehr wichtig, da es deutlich macht, dass mit der Definition eines *nichtdeterministischen* gegenüber der des *deterministischen* endlichen Automaten, keine größere Klasse von definierbaren Sprachen entsteht.

**3.24 Theorem**

*Jede von einem NFA akzeptierte Menge kann auch von einem initial zusammenhängenden, vollständigen DFA akzeptiert werden und ist daher regulär.*

Um dieses erstaunliche Theorem auf einfache Weise beweisen zu können ist es hilfreich, vorher spezielle Umformungen an einem vorgegebenen beliebigen, nichtdeterministischen endlichen Automaten vorzunehmen. So kann ein NFA eine oder mehrere $\lambda$-Kanten besitzen, die in deterministischen endlichen Automaten ja gar nicht erlaubt waren. Man kann diese nicht einfach entfernen, ohne die akzeptierte Sprache zu verändern, aber man kann einen beliebigen NFA zunächst derart umformen, dass er danach immer noch die gleiche Sprache akzeptiert jedoch keine $\lambda$-Kanten mehr besitzt. Wir nennen einen NFA ohne $\lambda$-Kanten nach Definition 3.10 kurz $\lambda$-frei.

**3.25 Theorem**

*Zu jedem NFA gibt es einen äquivalenten $\lambda$-freien NFA.*

**Beweis:** Sei $A := (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$ ein gegebener NFA, der möglicherweise $\lambda$-Kanten besitzt. Ein neuer NFA $B := (Z, \Sigma, K', Z_{\text{start}}, Z'_{\text{end}})$ ohne $\lambda$-Kanten wird wie folgt erklärt:

$$\begin{aligned} K' \quad := \quad & \{(z, w, z''') \in Z \times \Sigma^* \times Z \mid \exists (z', w, z'') \in K : \\ & w \neq \lambda \wedge z \xrightarrow[\lambda]{*} z' \wedge z'' \xrightarrow[\lambda]{*} z''' mbox(inA)\} \end{aligned}$$

und $Z'_{\text{end}} := Z_{\text{end}} \cup \{z \in Z_{\text{start}} \mid \exists z' \in Z_{\text{end}} : z \xrightarrow[\lambda]{*} z'\}$.

Wir zeigen die Gleichheit von$L(A)$ und $L(B)$ durch den Beweis der gegenseitigen Inklusion:

$L(B) \subseteq L(A)$ gilt, weil $(z, w, z')$ nur dann in $K'$ ist, wenn im Zustandsgraphen von $A$ ein Pfad von $z$ nach $z'$ existiert, auf dem das Wort $w$ gelesen wird. Die Definition von $Z'_{\text{end}}$ sichert, dass jeder Erfolgspfad in $B$ auch einer in $A$ ist.

$L(A) \subseteq L(B)$ gilt aus aus folgender Überlegung: Zunächst folgt aus $\lambda \in L(A)$ auch $\lambda \in L(B)$, denn dann ist einer der Startzustände von $B$ auf Grund der Definition von $Z'_{\text{end}}$ auch zugleich Endzustand.

Weiter ist klar, dass jede Kante $(z', w, z'') \in K$ mit $w \neq \lambda$ auch eine Kante von $K'$ ist!

Nun lässt sich jeder Erfolgspfad $p$ in $A$ schreiben als:

$$z_1 \xrightarrow[\lambda]{*} z'_1 \xrightarrow[w_1]{} z_2 \xrightarrow[\lambda]{*} z'_2 \xrightarrow[w_2]{} z_3 \xrightarrow[\lambda]{*} \cdots \xrightarrow[w_{n-1}]{} z_n \xrightarrow[\lambda]{*} z'_n \xrightarrow[w_n]{} z_{n+1} \xrightarrow[\lambda]{*} z'_{n+1}$$

Hierbei ist $z'_i \xrightarrow[w_i]{} z_{i+1}$ gerade der Zustandsübergang mit einer Kante $(z'_i, w_i, z_{i+1})$ von $A$, mit $w_i \neq \lambda$. Wenigstens eine solche Kante muss in einem Pfad $p$ mit $|p| \neq \lambda$ ja vorkommen. Nach

Definition von $K'$ ist aber jetzt auch $(z_i w_i, z_{i+1}) \in K'$ für jedes $1 \leq i < n$ sowie $(z_n, w_n, z'_{n+1}) \in K'$. Folglich gilt in dem Automaten $B$ auch $z_1 \xrightarrow[|p|]{*} z'_{n+1}$, was wegen $z_1 \in Z_{\text{start}}$ und $z_{n+1} \in Z'_{\text{end}}$ offensichtlich ein Erfolgspfad in $B$ ist.

□

Obwohl der Beweis oben gegeben ist, können wir damit noch nicht ganz zufrieden sein, denn eigentlich wissen wir doch gar nicht, auf welche Weise wir den neuen, $\lambda$-freien Automaten nun im speziellen Fall *effektiv* konstruieren können. Für die Konstruktion der neuen Kantenmenge $K'$ ist es offensichtlich nötig, festzustellen, ob für je zwei beliebige Zustände $z$ und $z'$ die Beziehung $z \xrightarrow[\lambda]{*} z'$ gilt. Nun ist die Relation $\xrightarrow[\lambda]{*} \subseteq Z \times Z$ gerade die reflexive, transitive Hülle der Relation $\xrightarrow[\lambda]{} := \{(p,q) | (p, \lambda, q) \in K\}$, und letztere kann sofort aus dem Zustandsgraphen von $A$ abgelesen werden: es sind genau die $\lambda$-Kanten dieses Automaten.

Zur effektiven Bestimmung des reflexiven, transitiven Abschlusses $R^*$ einer endlichen zweistelligen Relation $R$ (vergl. Def.2.16), also der Menge aller Wege zwischen den Knoten des gerichteter Graphen $G(R)$, taugt die Darstellung nach Theorem 2.26 natürlich nichts, denn es wäre ja ein unendlicher Durchschnitt zu bilden. Nach Theorem 2.25 wissen wir, dass es ausreicht von der Relation $\xrightarrow[\lambda]{}$ nur endlich viele Kompositionen $\xrightarrow[\lambda]{k}$ zu bilden um deren transitive Hülle $\xrightarrow[\lambda]{+}$ zu erhalten. Die dort benutzte Oberschranke ist jedoch etwas groß, und es stellt sich die Frage, ab es nicht bessere Verfahren gibt.

In dem gerichteten Graphen $G(R)$ beschreibt die reflexive transitive Hülle $R^*$ von $R$ alle gerichteten Pfade in diesem Graphen. Die Lösung dieser Frage ist auch an anderen Stellen von großem Interesse, und daher wird in der Veranstaltung F3 das Verfahren von Warshall zur Bestimmung des reflexiven, transitiven Abschlusses $\xrightarrow{*}$ einer endlichen zweistelligen Relation $\longrightarrow \subseteq A \times A$ behandelt.

Wir benutzen an dieser Stelle eine konstruktive Variante von Theorem 2.25, die eine recht gute Komplexität besitzt.

Sei $R \subseteq A \times A$ eine endliche Relation auf der endlichen Menge $A, n := |A|$. (In der hier gesuchten Anwendung wird $R := \xrightarrow[\lambda]{}$ mit $A := Z$ sein). Mit den Definitionen von $R^{i+1} := R^i \cdot R$ und $R^0 := Id_A$ gilt nun

$$\begin{aligned} R^* &= \bigcup_{i \geq 0} R^i \\ &= \bigcup_{i=0}^{n-1} R^i = Id_A \cup R \cup R^2 \cup R^3 \cup \cdots \cup R^{n-1} \\ &= (Id_A \cup R)^{n-1}, \end{aligned}$$

wobei letztere Gleichung leicht mit Induktion zu verifizieren ist. Man beachte hierbei, dass mit $R^i$ hier **nicht** das $i$-fache kartesische Produkt gemeint wurde sondern die $i$-fache Komposition! Warum der Exponent nicht größer als $n-1$ gewählt werden muss sieht man so ein:

Es gilt doch $(x, y) \in R^i$ genau dann, wenn es in dem Graphen G(R) einen Pfad vom Knoten $x$ zum Knoten $y$ gibt. Wenn es aber solch eine Pfad gibt, so existiert immer auch einer mit höchsten $n =$

$|A|$ verschiedenen Knoten, also mit höchstens n-1 Kanten. (Alle Schleifen können weggelassen werden, wodurch sich jeder Pfad sukzessive (rekursiv) verkürzen lässt.)

Damit braucht zur Bestimmung der Relation $\xrightarrow[\lambda]{*}$ nur das Produkt $(Id_Z \cup \xrightarrow[\lambda]{})^{|Z|-1}$ gebildet werden!

Für diese Aufgabe gibt es eine sehr effiziente Methode, die linear in der Größe $|A|$ arbeitet! Diese und weitere Methoden zu Design und Analyse von Algorithmen werden später im Studium, z.B. in der Veranstaltung F3 („Berechenbarkeit und Komplexität") aber auch im Hauptstudium noch ausführlicher behandelt werden.

Bis jetzt können wir also zu jedem NFA einen äquivalenten $\lambda$-freien NFA konstruieren, aber dieser ist noch immer nicht deterministisch, ja noch nicht einmal buchstabierend.

**3.26 Theorem**
*Zu jedem $\lambda$-freien NFA gibt es einen äquivalenten buchstabierenden NFA.*

**Beweis:** Zu jeder Kante $k := (z, w, z') \in K$ des NFA $A := (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$ mit $|w| \geq 2$ werden $|w| - 1$ neue (Zwischen-)Zustände $z_{k_i}$ definiert. Die Kante $k = (z, w, z')$ mit $w = x_1 x_2 \cdots x_n$ wird dann ersetzt durch die Kanten der Menge

$$\{(z, x_1, z_{k_1}), (z_{k_1}, x_2, z_{k_2}), \ldots, (z_{k_{n-2}}, x_{n-1}, z_{k_{n-1}}), (z_{k_{n-1}}, x_n, z')\}.$$

□

Nach diesen Umformungen eines NFA erhält man also mit Sicherheit einen äquivalenten $\lambda$-freien, buchstabierenden NFA. Dieser braucht aber nicht vollständig zu sein, da in manchen Zuständen nicht für jedes Eingabesymbol $x \in X$ eine herausführende Kante existieren muss. Will man auch dieses gewährleisten, so fügt man einen weiteren Zustand $z_s$ als *Senke* hinzu, und ergänzt den Automaten durch vorher fehlende Kanten, die in diese Senke führen. Im Zustand $z_s$ gibt es für jedes Symbol $x \in X$ eine Kante $(z_s, x, z_s)$. Bei der Umformung in einen deterministischen Automaten mit der folgende Konstruktion des sogenannten Potenzautomaten wird aber ohnehin ein vollständiger äquivalenter DFA entstehen, so dass dieses Verfahren nicht detallierter dargestellt werden braucht.

Nach diesen Vorbereitungen ist es jetzt einfacher möglich, den Beweis von Theorem 3.24 zu formulieren:

**Beweis (von Theorem 3.24):** Sei $A := (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$ ein buchstabierender, nicht notwendig vollständiger, NFA. Wir konstruieren den sog. **Potenzautomaten** zu $A$, dessen Zustände als Namen gerade die Teilmengen von $Z$ erhalten, und der ein vollständiger deterministischer endlicher Automat ist. (Wer damit besser umgehen kann, mag sich die Zustandsmenge so vorstellen, als stünde die Teilmenge $Q \subseteq Z$ für den Zustand $z_Q$ einer Menge $\{z_P \mid P \subseteq Z\}$.) Dieser vDFA $B$ werde wie folgt erklärt: $B := (2^Z, X, \delta, z_0, Z'_{\text{end}})$ mit $Z'_{\text{end}} := \{M \in 2^Z \mid M \cap Z_{\text{end}} \neq \emptyset\}$ und $z_0 := Z_{\text{start}}$, d.h., die Menge der Startzustände von $A$ bildet nun den einzigen Startzustand $z_0$ von $B$. Die Überführungsabbildung $\delta$ ist für alle $M \in 2^Z$ und jedes $x \in \Sigma$ definiert durch:

$$\delta(M, x) := \bigcup_{z \in M} \{z' \in Z \mid (z, x, z') \in K\}.$$

Zu jeder akzeptierenden Rechnung von $B$ auf einem Wort $w$ gibt es wegen der Definition von $\delta$ ebenfalls einen Erfolgspfad von $A$ auf $w$. Andererseits findet sich auch für jeden Erfolgspfad von $A$ eine akzeptierenden Rechnung in $B$. Die in $A$ besuchten Zustände kommen dabei in den einzelnen Zuständen des Pfades von $B$, die ja selbst Teilmengen von $Z$ sind, in der gleichen Reihenfolge vor. □

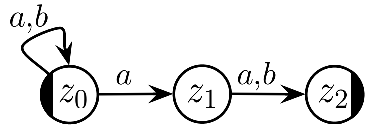



Abbildung 3.9: Der NFA akzeptiert alle Wörter, die an vorletzter Stelle ein $a$ haben

Der Potenzautomat „merkt“ sich im Zustandsnamen, in welchen Zuständen der nichtdeterministische endliche Automat sich nach derselben Sequenz von gelesenen Symbolen befinden könnte. Er akzeptiert ein Wort genau dann, wenn der ursprüngliche NFA mit diesem Wort in einnen Endzustand hätte gelangen können, d.h. genau dann, wenn es im ursprünglichen Automaten einen Erfolgspfad für dieses Wort gibt.

**3.27 Beispiel**

Wir konstruieren mit der Potenzautomatenkonstruktion zu dem in Abbildung 3.9 gezeichneten NFA $A := (Z, \Sigma, K, Z_{\mathsf{start}}, Z_{\mathsf{end}})$ einen äquivalenten vDFA $B$. Für den gegebenen NFA $A$ seien die Bestandteile erklärt durch:

$$\begin{array}{rcl}
Z & := & \{z_0, z_1, z_2\}, \\
\Sigma & := & \{a, b\}, \\
K & := & \{(z_0, a, z_0), (z_0, b, z_0), (z_0, a, z_1), (z_1, a, z_2), (z_1, b, z_2)\}, \\
Z_{\mathsf{start}} & := & \{z_0\}, \\
Z_{\mathsf{end}} & := & \{z_2\} \text{ und } .
\end{array}$$

Sein Zustandsgraph ist in Abbildung 3.9 dargestellt. Nach Konstruktion sind die Bestandteile von $B := (2^Z, X, \delta', z_0, Z'_{\mathsf{end}})$ gegeben durch:

$$\begin{array}{rcl}
Z & := & \Big\{\emptyset, \{z_0\}, \{z_1\}, \{z_2\}, \{z_0, z_1\}, \{z_0, z_2\}, \{z_1, z_2\}, \{z_0, z_1, z_2\}\Big\}, \\
\Sigma & := & \{a, b\}, \\
z_0 & := & \{z_0\} \\
Z'_{\mathsf{end}} & := & \Big\{\{z_2\}, \{z_0, z_2\}, \{z_1, z_2\}, \{z_0, z_1, z_2\}\Big\} \text{ und} \\
\delta & := & \text{entnehme man dem (nicht zusammenhängenden) Zustandsgraphen:}
\end{array}$$

Die graphische Darstellung des Potenzautomaten $B$ ist Abbildung 3.10 zu entnehmen.

Wie man in Abbildung 3.10 sehen kann, muss der Potenzautomat nicht initial zusammenhängend sein. Der rechte Teil der Abbildung zeigt einen vom Startzustand aus nicht erreichbaren Teilgraphen, der folglich für die Akzeptierung von Wörtern nicht relevant ist.

Natürlich kann man getrost alle diejenigen Komponenten eines vDFA weglassen, die nicht vom Startzustand aus erreicht werden können. Entsprechend muss dann die Übergangsfunktion auf die, eventuell kleinere, neue Zustandsmenge eingeschränkt werden. An der endlichen Darstellung des Zustandsgraphen kann man dies sofort sehen, falls der vDFA eine überblickbare Größe besitzt. Wir wollen hier

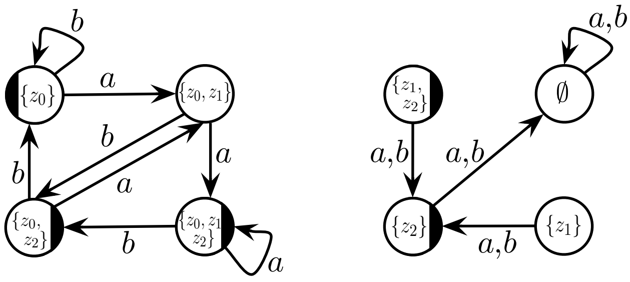



Abbildung 3.10: Potenzautomat zu dem NFA aus Abbildung 3.9

ein einfaches, wenn auch nicht das effizienteste, Verfahren vorstellen, wie man in dem Zustandsgraphen eines, nicht notwendig vollständigen, DFA die initial zusammenhängende Komponente bestimmen kann. Wenn wir in Zukunft vom Potenzautomaten sprechen, meinen wir üblicherweise nur den initial zusammenhängenden Teil des Potenzautomaten nach dem Beweis von Theorem 3.24.

**3.28 Algorithmus (initiale Zusammenhangskomponente eines vDFA)**

Sei $A := (Z, \Sigma, \delta, z_0, Z_{\mathsf{end}})$ ein beliebiger DFA. Wir berechnen schrittweise die induktiv definierten Mengen $M_i \subseteq Z$, beginnend bei $M_0$, gemäß:

$$M_i := \begin{cases} M_{i-1} \cup \bigcup\limits_{\substack{z \in M_{i-1} \\ x \in \Sigma}} \delta(z, x) & \text{für } i > 0 \\ \{z_0\} & \text{für } i = 0 \end{cases}$$

Offensichtlich ist jeder Zustand $z \in M_i$, $i \in I\!N$, mit einer passenden Eingabe $w \in \Sigma^*$, mit $|w| \leq i$, vom Startzustand aus erreichbar. Wegen $M_{i-1} \subseteq M_i \subseteq Z$ und der Endlichkeit der Zustandsmenge $Z$ muss es einen Index $k \leq |Z|$ geben, für den $M_{k+1} = M_k$ ist (mit jeder Erhöhung des Indexes $i$ um 1 kann möglicherweise nur ein weiterer Zustand in die Menge $M_i$ aufgenommen werden).

Das Konstruktionsverfahren endet, wenn das erste mal $M_k = M_{k+1}$ wird, was spätestens bei $k = |Z| - 1$ der Fall sein wird. Die Menge $M_k$ enthält bei Abschluss des Verfahrens also genau die von $z_0$ aus erreichbaren Zustände im vDFA $A$.

Nach der gegebenen Konstruktion des Potenzautomaten, hat der äquivalente DFA exponentiell mehr Zustände als der zu Beginn bekannte NFA. Dass dies keine Schwäche dieser Konstruktion ist, sondern in einigen Fällen tatsächlich soviele Zustände für den DFA nötig sind, zeigt das folgende Resultat von Theorem 3.29.

**3.29 Theorem**

*Zu jeder natürlichen Zahl $n \geq 2$ gibt es einen buchstabierenden, $\lambda$-freien NFA $A_n$ mit genau $n$ Zuständen, so dass* jeder *äquivalente vollständige DFA mindestens $2^n - 1$ Zustände besitzt.*

Die Automaten $A_n$ haben die Gestalt aus Abbildung 3.11.

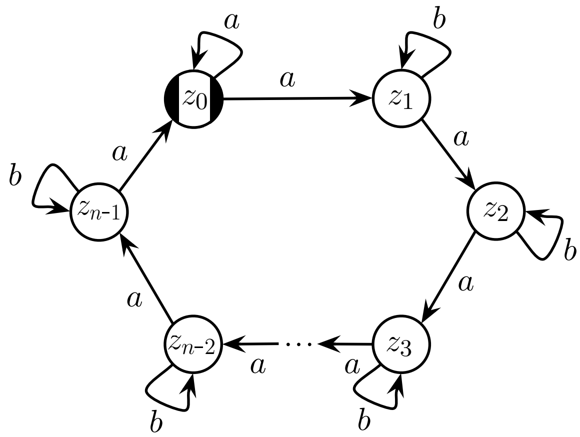



Abbildung 3.11: Der Automat $A_n$ hat genau $n$ Zustände.

Den Beweis dieser Aussage findet man als Übungsaufgabe in [Floyd&Beigel]. Wir werden später mit weniger Aufwand das ähnliche Theorem 3.50 beweisen.

Statt eines Beweises überlegen Sie bitte, welche Menge solch ein NFA $A_n$ beschreibt! An den Schwierigkeiten, diese Menge mit Worten exakt zu beschreiben, merken wir, dass eine andere Methode für die Beschreibung regulärer Mengen praktisch sein kann. In der Literatur findet man die Definition regulärer Mengen nicht nur durch endliche Automaten, sondern auch mit Hilfe von **rationalen Ausdrücken**. Rationale Ausdrücke werden deshalb auch oft als **reguläre Ausdrücke** bezeichnet. Den formalen Beweis für Äquivalenz zu endlichen Automaten wird Theorem 3.34 geben.

## 3.4 Rationale Ausdrücke

Im Folgenden wird nun ein weiteres, besonders typisches Beispiel einer *induktiven Definition* gegeben, bei der neu zu definierende Objekte durch schon bekannte erklärt werden (Definition der Syntax) und parallel dazu auch die Bedeutung (Semantik) definiert wird.

**3.30 Definition**

*Sei $X$ ein endliches Alphabet, dann sind* ***rationale Ausdrücke*** *über $X$ genau die mit den folgenden Regeln erzeugbaren Gebilde:*

1. *$\emptyset$ ist ein rationaler Ausdruck, der die (*leere*) Menge $M_\emptyset := \emptyset$ beschreibt.*
2. *Für jedes $a \in \Sigma$ ist $a$ ein rationaler Ausdruck, der die Menge $M_a := \{a\}$ beschreibt.*
3. *Sind $A$ und $B$ rationale Ausdrücke, welche die Mengen $M_A$ und $M_B$ beschreiben, dann ist auch*

    *$(A + B)$ ein rationaler Ausdruck, der die Menge $M_A \cup M_B$ beschreibt,*

    *$A \cdot B$ ein rationaler Ausdruck, der die Menge $M_A \cdot M_B$ beschreibt, und*

    *$(A)^*$ ein rationaler Ausdruck, der die Menge $M_A^*$ beschreibt.*

*$(A)^+$ ein rationaler Ausdruck, der die Menge $M_A^+$ beschreibt.*

4. *Nur die mit den Regeln 1.) bis 3.) erzeugbaren Gebilde sind rationale Ausdrücke.*

*Es bezeichne $\mathcal{R}at(\Sigma)$ die Familie aller mit rationalen Ausdrücken beschreibbaren Teilmengen von $\Sigma^*$, und $\mathcal{R}at := \bigcup_{\Sigma \text{ ist endl. Alphabet}} \mathcal{R}at(\Sigma))$.*

Um Klammern zu sparen, dürfen neben den äußersten Klammern auch weitere weggelassen werden, wobei die Prioritätenreihenfolge $()^*$ vor $\cdot$, und $\cdot$ vor $+$ für Eindeutigkeit sorgt. Damit steht $a \cdot b^* + a$ eindeutig für $(a \cdot (b)^* + a)$ und anstelle von $(c + (a + b))$ notieren wir $(a + b + c)$ als Beschreibung der Menge $\{a, b, c\}$. Es gelten die Assoziativgesetze für $+$ und $\cdot$ genauso wie die für die Vereinigung oder das Komplexprodukt, weshalb wir diese Klammern und den Punkt $\cdot$ meist ebenfalls weglassen können! Ebenfalls behandeln wir rationale Ausdrücke grundsätzlich wie Namen, bzw. Bezeichnungen der Mengen die sie beschreiben. Wir erlauben folgerichtig Notationen wie $babaa \in (a, b)^*$ anstelle von $babaa \in M_{(a,b)^*}$. In einigen Büchern wird in rationalen Ausdrücken auch $\Lambda$ als Beschreibung der Menge $\{\lambda\}$ benutzt. Dass dies hier nicht geschieht, ist kein entscheidender Verlust, denn $\emptyset^*$ beschreibt die gleiche Menge!

Rationale Ausdrücke werden in vielen Programmen zur Datei- oder Textverarbeitung benutzt, um Suchverfahren flexibler als nur mit sogenannten „*wildcards*“ vornehmen zu können. Manche Betriebssysteme tun dies auf der Ebene der Dateiverwaltung und ebenfalls einige Textverarbeitungsprogramme. Im UNIX-Betriebssystem kennt man zum Beispiel das Programm **egrep** (*__e__xtended __g__lobal __r__egular __e__xpression __p__rint*), was wir hier nur knapp und beispielhaft vorstellen wollen. Wer schon einmal mit dem UNIX-System gearbeitet hat, kennt vielleicht auch die Varianten „grep“ und „fgrep“.

Die Kommandosyntax für egrep ist:

$$\texttt{egrep}\ \langle regausdruck \rangle \langle dateiname \rangle.$$

Die Befehle zur Beschreibung der rationalen Ausdrücke müssen zwangsläufig anders dargestellt werden, und es gilt dabei:

| rationaler Ausdruck | egrep-Notation |
|---|---|
| Zeilenanfang | ˆ |
| Zeilenende | \$ |
| c | c |
| (Gesamtalphabet:) $\Sigma$ | . |
| $r + \emptyset^*$ | $r?$ |
| $r^*$ | $r*$ |
| $r^+$ | $r+$ |
| $r + s$ | $r \mid s$ |
| $r \cdot s$ | $r\ s$ |
| $(r)$ | $(r)$ |

Wenn man z.B. den Befehl

```
egrep  ^Th...i.$  /usr/dict/duden.
```

eingibt, so würden die Wörter `Thermik`, `Theorie` und `Thespis` gefunden und ausgedruckt. Wir wollen auf weitere Details und Feinheiten dieser Syntax hier nicht eingehen!

Für die Beschreibung von regulären Mengen scheinen sich die rationalen Ausdrücke geradezu anzubieten. So können wir z.B. die Menge $L_{a??}$ aller Wörter $w \in \{a, b\}^*$, bei denen das Symbol $a$ an drittletzter Position vorkommt einfach beschreiben: $L_{a??}$ wird durch $(a, b)^* a(a, b)(a, b)$ oder knapper durch $(a, b)^* a(a, b)^2$ beschrieben.

Wir werden im folgenden Abschnitt zeigen, dass es sich tatsächlich bei den Operationen, welche in der Definition der rationalen Ausdrücke verwendet werden, um Operationen handelt, gegen die die Familie der von endlichen Automaten abgeschlossen ist. Weitere Abschlusseigenschaften werden wir im Abschnitt 3.9 kennenlernen.

## 3.5 Einfache Abschlusseigenschaften

Es ist einfach zu sehen, dass eine Menge mit nur einem einzelnen Wort als Element und mithin auch jede endliche Menge von Wörtern durch einen rationalen Ausdruck beschrieben werden kann. Wenn wir nun beweisen könnten, dass die Vereinigung, das Produkt und die Sternbildung (Kleenesche Hülle) regulärer Mengen wieder reguläre Mengen sind, so wüßten wir, dass jeder rationale Ausdruck eine reguläre Menge beschreibt, die alternativ auch von einem endlichen Automaten akzeptiert werden könnte.

**3.31 Theorem**
*Mit $L_1 \in \mathcal{A}kz(\Sigma_1)$ und $L_2 \in \mathcal{A}kz(\Sigma_2)$ ist auch $L_1 \cup L_2 \in \mathcal{A}kz(\Sigma_1 \cup \Sigma_2)$.*

**Beweis (Konstruktion eines NFA für $L_1 \cup L_2$):** Seien zwei vDFA's $A := (Z_1, \Sigma_1, \delta_1, z_{1,0}, Z_{1,\text{end}})$ und $B := (Z_2, \Sigma_2, \delta_2, z_{2,0}, Z_{2,\text{end}})$ mit disjunkten Zustandsmengen gegeben, für die gilt $L_1 = L(A)$ und $L_2 = L(B)$.

Der NFA $C_{A \cup B} := (Z_3, \Sigma_3, K_3, z_{3,0}, Z_{3,\text{end}})$ für $L_1 \cup L_2$ ist gegeben durch:

$$
\begin{aligned}
Z_3 &:= Z_1 \uplus Z_2, \\
\Sigma_3 &:= \Sigma_1 \cup \Sigma_2, \\
Z_{3,\text{start}} &:= \{z_{1,0}, z_{2,0}\}, \\
Z_{3,\text{end}} &:= Z_{1,\text{end}} \cup Z_{2,\text{end}} \text{ und} \\
K_3 &:= \{(p_1, x, \delta_1(p_1, x)) \mid p_1 \in Z_1, x \in \Sigma_1\} \cup \{(p_2, x, \delta_2(p_2, x)) \mid p_2 \in Z_2, x \in \Sigma_2\}.
\end{aligned}
$$

Dieser Automat ist zwar nichtdeterministisch und im Allgemeinen nicht vollständig, aber schon buchstabierend. Wie es möglich ist, daraus einen äquivalqnten vDFA zu konstruieren haben wir mit der Potenzautomatenkonstruktion von Theorem 3.24 schon gesehen. Es ist offensichtlich, dass zu jeder Erfolgsrechnung in $A$ (bzw in $B$) eine entsprechende Erfolgsrechnung in $C_{A \cup B}$ existiert, denn mit der Wahl des Startzustandes wird entschieden, ob ein Wort der Menge $L_1$ oder $L_2$ akzeptiert werden soll. Umgekehrt, entspricht einer Erfolgsrechnung in $C_{A \cup B}$, die in einem der Endzustände von $A$ (bzw. von $B$) endet, die entsprechende Erfolgsrechnung in $A$ (bzw. in $B$). □

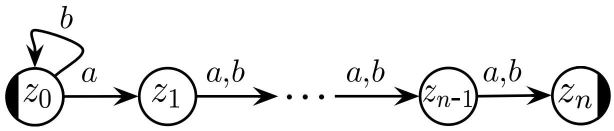



Abbildung 3.12: Ein NFA für die Sprache $\{b\}^* \cdot \{a\} \cdot \{a,b\}^{n-1}$

**3.32 Theorem**
*Mit $L_1 \in \mathcal{R}at(\Sigma_1)$ und $L_2 \in \mathcal{R}at(\Sigma_2)$ ist auch $L_1 \cdot L_2 \in \mathcal{R}at(\Sigma_1 \cup \Sigma_2)$.*

**Beweis (Konstruktion eines buchstabierenden NFA's für $L_1 \cdot L_2$):** Seien $A := (Z_1, \Sigma_1, \delta_1, z_{1,0}, Z_{1,\text{end}})$ und $B := (Z_2, \Sigma_2, \delta_2, z_{2,0}, Z_{2,\text{end}})$ vollständige DFA's mit $Z_1 \cap Z_2 = \emptyset$ und $L_1 = L(A)$ und $L_2 = L(B)$. Ein buchstabierender NFA $C_{A \cdot B}$ für das Produkt $L_1 \cdot L_2$ ist $C_{A \cdot B} := (Z_1 \cup Z_2, \Sigma_1 \cup \Sigma_2, K, z_{1,0}, Z_{2,\text{end}})$ mit $K := K_1 \cup K_2 \cup K_3$ wobei $K_1 := \{(p_1, x, \delta_1(p_1, x)) \mid x \in \Sigma_1, p_1 \in Z_1\}$ die zum vDFA $A$ gehörende Kantenmenge, $K_2 := \{(p_2, x, \delta_2(p_2, x)) \mid x \in \Sigma_2, p_2 \in Z_2\}$ die zum vDFA $B$ gehörende Kantenmenge und $K_3 := \{(p_1, x, \delta_2(z_{2,0}, x)) \mid x \in \Sigma_2, p_1 \in Z_{1,\text{end}}\}$ die beide Automaten verbindenden, $\lambda$-freien Kanten sind. □

Bei obiger Konstruktion wird der NFA $C_{A \cdot B}$ in der Regel kein vollständigter DFA sein, so z.B. wenn $\Sigma_1 \cap \Sigma_2 = \emptyset$ gilt. Ist $\Sigma_1 = \Sigma_2$, so ist es leider zudem nötig, den entstandenen NFA in einen deterministischen umzuformen. Es kann dabei sogar geschehen, dass für vDFA's $A$ und $B$ mit $|Z_1| = m$ und $|Z_2| = n$ jeder äquivalente DFA für $L(A) \cdot L(B)$ mindestens $2^{m+n}$ Zustände haben muss! Als illustratives, aber diese Aussage nicht beweisendes, Beispiel betrachten wir den Automaten $A := (\{z_0\}, \{a,b\}, \delta, z_0, \{z_0\})$ mit $\delta(z_0, a) := \delta(z_0, b) := z_0$ und $L(A) = \{a,b\}^*$. Für $B$ wählen wir den Automaten Abbildung 3.12, mit $L(B) = \{b\}^* \cdot \{a\} \cdot \{a,b\}^{n-1}$. Es ergibt sich $L(A) \cdot L(B) = \{a,b\}^* \cdot \{a\} \cdot \{a,b\}^{n-1}$. Von dieser Menge werden wir nach dem Beweis von Theorem 3.50 wissen, dass sie kein vDFA mit weniger als $2^n$ Zuständen akzeptiert!

Wenn wir für die Sternbildung ebenfalls einen DFA konstruieren müssen, so gelingt dies auf einfache Weise auch nicht ohne die übliche Umformung eines buchstabierenden NFA in den äquivalenten DFA:

**3.33 Theorem**
*Mit $L \in \mathcal{A}kz(\Sigma)$ ist auch $L^* \in \mathcal{A}kz(\Sigma)$.*

**Beweis (Konstruktion eines buchstabierenden NFA's für $L^*$):** Sei $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein vDFA mit $L = L(A)$, so wird ein buchstabierender NFA $C_{A^*}$ definiert durch $C_{A^*} := (Z \uplus \{p\}, X, K, \{p\}, \{p\})$ mit $K := K_1 \cup K_2 \cup K_3$, wobei $K_1 := \{(p, x, \delta(p, x)) \mid x \in X, p \in Z\}$ die zum vDFA $A$ gehörende Kantenmenge, sowie $K_2 := \{(p, x, \delta'(z_0, x)) \mid x \in X\}$ und $K_3 := \{(q, x, p) \mid x \in X, \delta'(q, x) \in Z_{\text{end}}\}$ die $\lambda$-freien Verbindungen mit dem neuen Zustand sind. □

Wir werden im folgenden Theorem zeigen, dass auch umgekehrt *jede* reguläre Menge durch einen rationalen Ausdruck beschrieben werden kann. Daher unterscheidet sich die Familie aller Sprachen, die durch rationale Ausdrücke definierbar sind, nicht von denen, die durch endliche Automaten akzeptiert werden können.

**3.34 Theorem (Satz von Kleene, 1951)**
*Für jedes Alphabet X gilt* $\mathcal{A}kz(\Sigma) = \mathcal{R}at(\Sigma)$ *und folglich* $\mathcal{R}eg = \mathcal{R}at$.

**Beweis:** Der Beweis erfolgt wiederum in zwei Schritten, in denen die Inklusion einer der beiden Familien in der jeweils anderen gezeigt wird.

$\mathcal{R}at(\Sigma) \supseteq \mathcal{A}kz(\Sigma)$**:** Wir definieren zu jedem vollständigen DFA $A = (Z, \Sigma, \delta, z_1, Z_{\text{end}})$ die reguläre Menge $L(A)$, zu der ein rationaler Ausdruckk leicht gebildet werden kann. Dazu seien für alle $0 \leq k \leq |Z|$ und alle $1 \leq i,\ j \leq |Z|$ mit $Z := \{z_1, \ldots, z_n\}$ folgende Mengen definiert:

$$R^k_{i,j} := \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} \delta(z_i, w) = z_j \text{ und für } w = uv \text{ mit } u \neq \lambda \neq v \\ \text{folgt aus } \delta(z_i, u) = z_r \text{ stets } r \leq k \end{array} \right\}.$$

Die Menge $R^k_{i,j}$ enthält alle Wörter, die auf Pfaden vom Zustand $z_i$ zum Zustand $z_j$ über die Menge $\{z_1, \ldots, z_k\}$ von Zwischenzuständen gelesen werden können. Folglich ist

$$R^0_{i,j} = \{x \in X \cup \{\lambda\} \mid \delta(z_i, x) = z_j \text{ oder } (x = \lambda) \wedge (i = j)\}.$$

Wir zeigen nun die Gültigkeit der folgenden Gleichung:

$$R^k_{i,j} = \begin{cases} \{x \mid \delta(z_i, x) = z_j \text{ oder } (x = \lambda) \wedge (i = j)\}, & \text{falls } k = 0 \\ R^{k-1}_{i,j} \cup R^{k-1}_{i,k} \cdot (R^{k-1}_{k,k})^* \cdot R^{k-1}_{k,j}, & \text{falls } k \geq 1. \end{cases}$$

**Induktions-Basis:** $R^0_{i,j}$ ist in der Definition wie in der ersten der zwei Gleichungen angegeben.

**Induktions-Schritt:** Für $k = m \geq 0$ sei für die Mengen $R^m_{i,j}$ die Rekursionsformel bewiesen. und wir zeigen, dass sie dann auch für die Mengen $R^{m+1}_{i,j}$ gilt. Jeder Pfad von $z_i$ nach $z_j$, der nur Knoten der Menge $\{z_1, \ldots, z_m, z_{m+1}\}$ durchläuft, kann in Stücke zerlegt werden, in denen nur Knoten der Menge $\{z_1, \ldots, z_m\}$ durchlaufen werden, die aber stets beim Knoten $z_{m+1}$ aneinander gesetzt sind, sofern dieser überhaupt vorkommt. Kommt $z_{m+1}$ nicht vor, so gehört das auf diesem Pfad gelesene Wort zur Menge $R^m_{i,j}$. Kommt $z_{m+1}$ wenigstens einmal auf dem Pfad vor, so hat dieser eine Zerlegung in der folgenden Art:

$$z_i \xrightarrow[u]{*} z_{m+1} \xrightarrow[v_1]{*} z_{m+1} \xrightarrow[v_2]{*} z_{m+1} \xrightarrow[v_3]{*} \cdots \xrightarrow[v_r]{*} z_{m+1} \xrightarrow[w]{*} z_j$$

Die Pfade mit den Wörtern $u, v_1, v_2, \ldots, v_r, w$ durchlaufen nun nur noch Zustände der Menge $\{z_1, \ldots, z_m\}$.

Offensichtlich gilt: $u \in R^m_{i,m+1}, \forall 1 \leq i \leq r:\ v_i \in R^m_{m+1,m+1}$ und $w \in R^m_{m+1,j}$. Dies zeigt zunächst die Inklusion: $R^{m+1}_{i,j} \subseteq R^m_{i,j} \cup R^m_{i,m+1} \cdot (R^m_{m+1,m+1})^* \cdot R^m_{m+1,j}$. Die Umkehrung dieser Inklusion ist leicht ersichtlich.

Für die oben definierten Mengen $R^k_{i,j}$ gilt nun offensichtlich $L(A) = \bigcup\limits_{z_j \in Z_{\text{end}}} R^n_{1,j}$, wobei $z_1$ der Startzustand des Automaten $A$ war.

Setzt man nun die Rekursionsformel solange in $R^n_{1,j}$ ein, bis die Anfangsmengen $R^0_{i,j}$ erreicht sind, so erhält man eine endliche Beschreibung der Menge $L(A)$, die leicht in einen rationalen Ausdruck umgeschrieben werden kann.

$\mathcal{R}at(\Sigma) \subseteq \mathcal{A}kz(\Sigma)$**:** Wir benutzen die Induktive Definition der rationalen Ausdrücke und zeigen zuerst, dass die einfachen Basismengen durch endliche Automaten dargestellt werden können. Dann werden schon bekannte Konstruktionen an NFA's entsprechend der Zusammenfügung schon bestehender rationaler Ausdrücke zu Neuen durchgeführt:

1. $\emptyset$ ist ein rationaler Ausdruck, der die leere Menge beschreibt und wird von dem DFA $C_\emptyset := (\{z_0\}, \Sigma, \delta, z_0, \emptyset)$ akzeptiert. ($C_\emptyset$ hat ein beliebiges Eingabealphabet, jedoch keinen Endzustand).
2. Für jedes $a \in \Sigma$ ist $a$ ein rationaler Ausdruck, der die Menge $\{a\}$ beschreibt und diese wird von dem DFA $C_a := (\{z_0, z_1\}, \{a\}, \delta, z_0, \{z_1\})$ mit $\delta(z_0, a) := z_1$ akzeptiert.
3. Sind $A$ und $B$ rationale Ausdrücke, und $C_A := (Z_A, X_A, K_A, Z_{A,\text{start}}, Z_{A,\text{end}})$ sowie $C_B := (Z_B, X_B, K_B, Z_{B,\text{start}}, Z_{B,\text{end}})$ NFA's mitdisjunkten Zustandsmengen, welche die durch die rationalen Ausdrücke dargestellten Mengen $M_A$ und $M_B$ akzeptieren, dann akzeptiert der NFA
   a) $C_{A\cup B}$ von Theorem 3.31 die durch $(A + B)$ beschriebene Menge $M_A \cup M_B$.
   b) $C_{A\cdot B}$ von Theorem 3.32 die durch $A \cdot B$ beschriebene Menge $M_A \cdot M_B$.
   c) $C_{A^*}$ von Theorem 3.33 die durch den rationalen Ausdruck $(A)^*$ beschriebene Menge $M_A^*$.
   d) $C_{A^+} := (Z_A, X_A, K_A \cup \{(p, \lambda, q) \mid p \in Z_{A,\text{end}}, q \in Z_{A,\text{start}}\}, Z_{A,\text{start}}, Z_{A,\text{end}})$ die durch den rationalen Ausdruck $(A)^+$ beschriebene Menge $M_A^+$.
4. Damit kann für jeden rationalen Ausdruck $r$ ein endlicher Automat konstruiert werden, der die von diesem beschriebene Menge $M_r$ akzeptiert.

Hiermit ist der Beweis von Theorem 3.34 abgeschlossen, denn wir haben gezeigt, wie für einen beliebigen NFA ein äquivalenter rationaler Ausdruck erzeugt werden kann und dass umgekehrt zu jedem rationalen Ausdruck auch ein äquivalenter endlicher Automat existiert. □

Wir können nun die rationalen Ausdrücke als einfache syntaktische Darstellung von regulären Mengen benutzen, ohne sofort endliche Automaten konstruieren zu müssen. Wir könnten sogar ohne weitere Nachteile die Syntax der rationalen Ausdrücke um solch Notationen erweitern wie: $(A)^n$ für jedes $n \in \mathbb{N}$, $(A \cap B)$ oder $\overline{(A)}$. Wir werden im Folgenden solche erweiterten rationalen Ausdrücke freizügig benutzen!

## 3.6 Ein Anwendungsbeispiel

Es folgt nun ein etwas größeres Anwendungsbeispiel für die Nutzung von endlichen Automaten mit Ausgabe, welche verbunden wird mit einer rekursiven Definition der benötigten Zustände. Wir wollen das Problem der Suche und Identifizierung von endlich vielen Schlüsselwörtern in einem Text mit Hilfe eines endlichen (Moore-)Automaten lösen. Die hier gewählte Darstellung unterscheidet sich von den in Lehrbüchern üblichen dadurch, dass wir auf eine mögliche Implementierung, d.h. einer Umsetzung und Verwirklichung in einem realen Programm, weniger Wert legen als auf die prinzipielle Struktur. Unsere Lösung ist eine rekursive Variante des Verfahrens von Donald E. Knuth, James H. Morris und Vaughn R. Pratt (1977).

**Problemstellung:** Ein fortlaufender Text soll daraufhin untersucht werden, ob gewisse „Schlüsselwörter“ (*keys*) darin enthalten sind, und ausgegeben werden, um welche es sich handelt und an welchen Stellen diese im Text auftreten. Diese Schlüsselwortsuche soll natürlich auf beliebige Texte angewendet und ebenfalls auch mit wechselnden Schlüsselwörtern erneut gestartet werden können. Das bedeutet, dass als Eingabe des zu entwerfenden Algorithmusses nur die endliche Menge $K \subseteq \Sigma^*$ der Schlüsselwörter verwendet werden kann, nicht aber der Text $t \in \Sigma^*$, auf den das Verfahren ja angewendet werden soll.

Ob in einem Text $t \in \Sigma^*$ überhaupt eines der Schlüsselwörter der Menge $K \in \Sigma^*$ vorkommt, könnte man feststellen, in dem man die Frage „$\{t\} \cap \Sigma^* \cdot K \cdot \Sigma^* \neq \emptyset$?“ beantwortet. Da die Mengen $\{t\}$ und $\Sigma^* \cdot K \cdot \Sigma^*$ offensichtlich regulär sind, könnte man mit Hilfe des Beweises von Theorem 3.34 eine NFA konstruieren, der diese Menge akzeptiert. Für diesen NFA müsste mann dann noch feststellen, ob dieser die leere Menge akzeptiert oder nicht. Niemand würde wirklich in dieser Weise vorgehen, da der Zeitaufwand immens wäre! Außerdem würden wir auf diese Weise nicht feststellen können, an welchen Stellen im Text $t$ welche Schlüsselwörter vorkommen und unser Aufgabe wäre nur unzureichend gelöst.

Für jede endliche Menge $K \subseteq \Sigma^*$ von Schlüsselwörtern konstruieren wir nun einen Moore-Automaten $A$ der $\Sigma^* \cdot K \cdot \Sigma^*$ mit den passenden Ausgaben versieht direkt:

### 3.35 Definition

*Sei $\Sigma$ ein Alphabet und $K \subseteq \Sigma^*$ eine endliche Menge von Schlüsselwörtern. Der* ***Moore-Automat*** *$A_K$* ***zur Schlüsselwortsuche*** *$A_K := (Z, \Sigma, \Gamma, \delta, \rho, z_\lambda)$ ist erklärt durch:*

$$Z := \{z_v \mid \exists w \in K : v \sqsubseteq w\},$$

$$\forall z_v \in Z : \forall x \in \Sigma : \delta(z_v, x) := \begin{cases} z_{vx} & \text{wenn } z_{vx} \in Z \text{ ist,} \\ \delta(z_u, x) & \text{wenn } z_{vx} \notin Z, \text{ und } v = yu, y \in \Sigma, \\ z_\lambda & \text{sonst.} \end{cases}$$

$$\rho(z_w) := \begin{cases} \text{„Schlüsselwörter } v_1, v, \ldots v_k \text{ wurden gefunden!“,} \\ \text{falls } \forall i \in \{1, \ldots, k\} : (v_i \in K) \wedge \exists\, u : w = uv_i \ (v_i \text{ ist suffix von } w). \\ \\ \lambda, \text{ d.h. keine Ausgabe, sonst .} \end{cases}$$

Wir wollen die Definitiion des Moore-Automaten zur Schlüsselwortsuche anhand eines Beispiels verdeutlichen.

### 3.36 Beispiel

Betrachten wir folgende Menge von Schlüsselwörtern. $K := \{der, erde\} \subseteq \{d, e, r\}^*$, so erhalten wir die Zustandsmenge $Z = \{z_\lambda, z_d, z_{de}, z_{der}, z_e, z_{er}, z_{erd}, z_{erde}\}$ und folgenden Moore-Automaten. Man kann zeigen, dass der hier definierte Moore-Automat zur Textsuche der kleinste mit dieser Eigenschaft ist. Er hat $|Z| = 1 + \sum_{v \in K} |v|$ viele Zustände und kann schnell in $c \cdot |\Sigma| \cdot |Z|$ Schritten konstruiert werden, in dem man die Kanten bei $z_\lambda$ beginnend bestimmt. Ein weiterer Vorteil dieses Automaten ist, dass er bei der Suche nach den Schlüsselwörtern jedes Zeichen des Textes $t$ nur genau einmal liest und ohne Rücksprünge in Realzeit arbeitet. Zudem kann dieses Verfahren leicht für ähnliche Probleme abgewandelt werden.

In Abbildung 3.13 sind die offensichtlichen Ausgaben in den Zuständen $z_{erde}$ und $z_{der}$ nicht eingezeichnet worden.

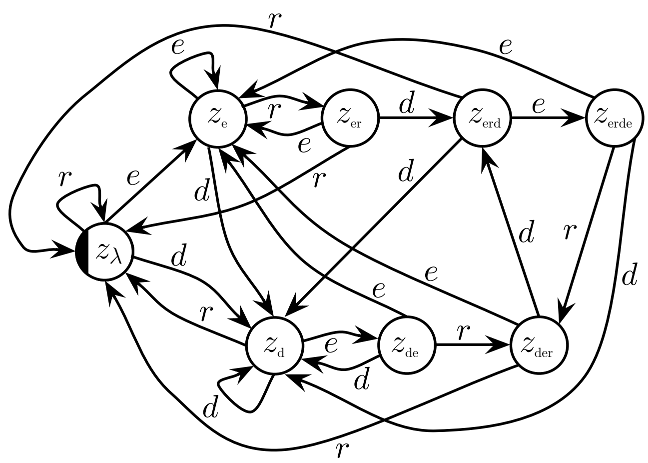



Abbildung 3.13: Ein Moore-Automat zur Schlüsselwortsuche für die Wörter „erde“ und „der“

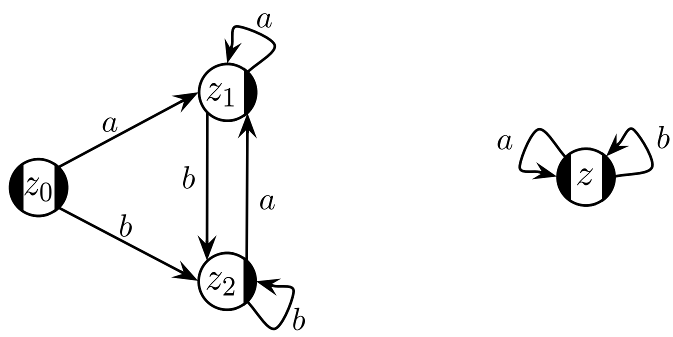



Abbildung 3.14: Zwei äquivalente Automaten

## 3.7 Äquivalenzbegriffe und minimale Automaten

Betrachtet man die beiden in Abbildung 3.14 abgebildeten vDFA's, so stellt sich schnell heraus, dass die beiden endlichen Automaten beide die gleiche Sprache akzeptieren, nämlich $\{a, b\}^*$.

Wir wollen jetzt daran gehen, die Frage zu beantworten, wie ein kleinster äquivalenter DFA aussieht, ob er immer konstruiert werden kann und ob es vielleicht davon unterschiedliche geben kann. Vorher erklären wir den Begriff „Transitionsmonoid“ und eine sehr praktische Notation.

Für einen vollständigen DFA $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ gibt es zu jedem Zustand $z \in Z$ und jedem Wort $w \in \Sigma^*$ genau einen Folgezustand $z'$ mit $z \xrightarrow[w]{*} z'$. Jedes $w \in \Sigma^*$ induziert für diesen Automaten also eine Abbildung: $f_w : Z \longrightarrow Z$ mit $f_w(z) := \delta(z, w)$.

**3.37 Definition**
*Zu einem gegebenen vDFA $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ schreiben wir $(z)^w$ für den Zustand $\delta(z, w)$, der auf eindeutige Weise von $z$ aus bei Eingabe von $w$ erreicht wird.*

*Die Menge $F_A := \{(\ )^w \mid w \in \Sigma^*\}$ der Abbildungen $(\ )^w : Z \longrightarrow Z$, mit denen die Zustandsübergange des vDFA's $A$ beschrieben werden, bilden zusammen mit der Komposition von Funktionen das endliche* ***Transitionsmonoid*** *$(F_A, \circ)$ zu $A$.*

Wieso ist die Menge $F_A$ aller Abbildungen $(z)^w : Z \longrightarrow Z$ für jeden vollständigen DFA $A$ endlich? Wieviele Abbildungen dieser Art gibt es?

Es gilt offensichtlich $((z)^u)^v = (z)^{uv}$, was die Komposition $(f_v \circ f_u)(z) = f_v(f_u(z))$ in der üblicheren Notation darstellt. Weiter ist $(z)^\lambda = z$ für jeden Zustand $z \in Z$, d.h. stets ist $f_\lambda$ neutrales Element des Transitionsmonoides.

Ein DFA $A$ definiert in natürlicher Weise eine rechtsinvariante Äquivalenzrelation $R_A$, deren Index der Zahl der nicht überflüssigen Zustände von $A$ gleicht. Wir wiederholen hier die aus der Mathematik bekannten Definitionen zur besseren Lesbarkeit.

**3.38 Definition**
*Eine Relation $R \subseteq A \times A$ heißt* ***Äquivalenzrelation*** *genau dann, wenn $R$* reflexiv, symmetrisch *und* transitiv *ist.*

*Ist $R \subseteq A \times A$ eine Äquivalenzrelation auf $A$, so heißt die Menge*

$$[a]_R := \{b \in A \mid (a, b) \in R\}$$

*die* ***Äquivalenzklasse*** *von $a$. Jedes $b \in [a]$ ist ein* ***Repräsentant*** *von $[a]_R$. Man schreibt gewöhnlich nur $[a]$, wenn deutlich ist, welche Relation gemeint ist.*

*Die Menge aller Äquivalenzklassen notieren wir als ${}^{A}\!/\!_{R} := \{[a]_R \mid a \in A\}$.*

*Als* ***Index*** *der Äquivalenzrelation $R$ auf $A$ bezeichnet man die Zahl der verschiedenen Äquivalenzklassen, die $R$ hat. Wir notieren dies durch: $Index(R) := \left|{}^{A}\!/\!_{R}\right|$.*

Eine Äquivalenzrelation $R$ auf einer Menge $A$ zerlegt diese in disjunkte Teilmengen und spezifiziert dadurch eine **Partition** von $A$.

Wir formulieren dies als Theorem:

**3.39 Theorem**
*Ist $R \subseteq A^2$ eine Äquivalenzrelation, so bilden die Äquivalenklassen bezüglich $R$ eine Partition von $A$.*

**Beweis:** Zunächst zeigen wir $A = \bigcup_{a \in A} [a]_R$. Sei dazu $x \in A$ beliebig, so gilt $xRx$ wegen der Reflexivität von $R$. Mithin gilt $x \in [x]_R$, erst recht also $x \in \bigcup_{a \in A} [a]_R$. Folglich gilt $A \subseteq \bigcup_{a \in A} [a]_R$. Umgekehrt ist $[a]_R \subseteq A$ auf Grund der Definition von $[a]_R$.

Zu zeigen ist nun noch, dass für alle $[a]_R, [b]_R \in {}^{A}\!/\!_{R}$ entweder $[a]_R = [b]_R$ oder $[a]_R \cap [b]_R = \emptyset$ gilt. Seien $a, b \in A$ mit $[a]_R \cap [b]_R \neq \emptyset$. Also muss es ein $c \in [a]_R \cap [b]_R$ geben, für das dann $aRc$ und $bRc$ gilt. Sei nun

$d \in [a]_R$ ein beliebiges Element, so gilt $dRa$, $aRc$ und $cRb$, wobei die Symmetrie von $R$ ausgenutzt wird. Aus der Transitivität von $R$ folgt nun $dRb$ und $d \in [b]_R$. Also ist $[a]_R \subseteq [b]_R$. Durch Vertauschen der Symbole $a$ und $b$ in den letzten beiden Sätzen gewinnt man mit der gleichen Begründung die Inklusion $[b]_R \subseteq [a]_R$, so dass aus $[a]_R \cap [b]_R \neq \emptyset$ tatsächlich $[a]_R = [b]_R$ folgt. □

Zwei Äquivalenzklassen einer Äquivalenzrelation sind also entweder identisch oder disjunkt! Speziell die Identität ist eine symmetrische, reflexive, und transitive Relation, also eine Äquivalenzrelation. Selbstverständlich ist die Identitätsrelation auch antisymmetrisch. Aber im Allgemeinen sind Äquivalenzrelationen gerade nicht antisymmetrisch!

**3.40 Beispiel**

Betrachten Sie die folgende Relation: $\equiv_4 \subseteq (\{0,1,2,3,4,5,6,7,8,9\})^2$ mit $a \equiv_4 b$ genau dann, wenn $b - a$ ein Vielfaches von $4$ ist. Dass $\equiv_4$ eine Äquivalenzrelation ist, kann leicht nachgeprüft werden. Aber sicherlich ist $\equiv_4$ nicht antisymmetrisch, denn es gilt $[1]_{\equiv_4} = [5]_{\equiv_4} = [9]_{\equiv_4}$, also $1 \equiv_4 5$ und $5 \equiv_4 1$ aber nicht $1 = 5$.

Oft werden Äquivalenzrelationen (genauso wie Kongruenzrelationen) mit dem (meist noch mit eienm Index versehenen) Symbol $\equiv$ bezeichnet, aber auch das mit $R$ indizierte Gleichheitszeichen $=_R$ ist nicht ungebräuchlich.

**3.41 Definition**

*Eine Äquivalenzrelation $\equiv$ hat* ***endlichen Index****, wenn die Zahl* $\mathrm{ind}(\equiv)$ *der Äquivalenzklassen endlich ist.*

*Eine Äquivalenzrelation $\equiv \subseteq M \times M$ heißt genau dann* ***rechtsinvariant****, wenn für jedes Element $m$ der Halbgruppe $(M, \circ)$ aus $m_1 \equiv m_2$ stets $(m_1 \circ m) \equiv (m_2 \circ m)$ folgt.*

Nach dieser wichtigen definition einer Äquivalenzrelation wollen wir einen kleinen Exkurs zu den Grenzen der regulären Sprachen unternehmen, um eine Anwendung dieses Zweiges der Theorie aufzuzeigen.

### 3.7.1 Exkurs: Grenzen der regulären Sprachen

Am Ende dieses Abschnitts werden wir eine Sprache kennenlernen, von der wir zeigen, dass sie nicht regulär ist. Dazu müssen wir noch einige Begriffe einführen.

**3.42 Definition**

*Sei $A = (Z, \Sigma, \delta, \{z_0\}, Z_{end})$ ein DFA, dann ist die Äquivalenzrelation $R_A \subseteq \Sigma^* \times \Sigma^*$ erklärt durch:*

$$u \, R_A \, v \text{ genau dann, wenn } (z_0)^u = (z_0)^v$$

*Sei $L \in \Sigma^*$ eine reguläre Menge, dann ist die* ***Nerode-Äquivalenz*** *oder* ***syntaktische Rechtskongruenz*** *$R_L \subseteq \Sigma^* \times \Sigma^*$ erklärt durch:*

$$u \, R_L \, v \quad \text{genau dann, wenn} \quad \forall w \in \Sigma^* : (uw \in L \quad \text{genau dann, wenn} \quad vw \in L).$$

Es ist leicht zu sehen, dass sowohl $R_A$ als auch $R_L$ eine rechtsinvariante Äquivalenzrelation ist.

Das folgende Ergebnis ist als **Satz von Nerode** bekannt und zeigt, dass $R_L$ von endlichem Index ist:

**3.43 Theorem (Myhill & Nerode, 1957/1958)**
*Die folgenden Aussagen sind äquivalent:*

1. *$L \subseteq \Sigma^*$ ist regulär.*
2. *$L$ ist Vereinigung von Äquivalenzklassen einer rechtsinvarianten Äquivalenzrelation mit endlichem Index.*
3. *Die Nerode-Äquivalenz $R_L$ hat endlichen Index.*

**Beweis:** Wir führen einen Ringschluss durch, indem wir zunächst zeigen, dass Aussage *2.* aus *1.* folgt. Dann schließen wir von *2.* auf *3.* und zeigen dann noch, dass *1.* aus *3.* folgt. Damit ist dann die Äquivalenz der drei Aussagen gezeigt.

**1. → 2.:** Sei $A = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein nach Theorem 3.24 existierender DFA, der $L \subseteq \Sigma^*$ akzeptiert, und $R_A$ die Relation auf $\Sigma^*$ gemäß Definition 3.42 mit: $uR_Av$ genau dann, wenn $(z_0)^u = (z_0)^v$. $L$ ist die Vereinigung derjenigen Klassen, die zu Endzuständen gehören, d.h. $L = \{w \in \Sigma^* \mid (z_0)^w \in Z_{\text{end}}\}$, und da das Transitionsmonoid endlich ist, hat auch $R_A$ nur endlich viele Klassen. $R_A$ ist rechtsinvariant, da im DFA die mit einem Wort $w \in \Sigma^*$ aus $z \in Z$ erreichten Zustände $(z)^w$ stets eindeutig sind.

**2. → 3.:** Sei $R$ die nach Voraussetzung existierende rechtsinvariante Äquivalenzrelation von endlichem Index. Wir zeigen, dass $R_L$ eine Vergröberung von $R$ ist, d.h. $xRy$ impliziert $xR_Ly$. Damit bestehen die Äquivalenzklassen bezüglich $R_L$ aus der Vereinigung einiger Äquivalenzklassen bezüglich $R$. Folglich gilt $\text{ind}(R_L) \leq \text{ind}(R)$ und die Nerode-Äquivalenz $R_L$ ist von endlichem Index. Sei also $xRy$. Wegen der Rechtsinvarianz von $R$ folgt $xzRyz$ für jedes $z \in \Sigma^*$. Da nach Voraussetzung jede Äquivalenzklasse von $R$ entweder ganz zu $L$ gehört oder kein Wort aus $L$ enthält, gehören entweder beide Wörter $xz$ und $yz$ zu $L$ oder beide nicht zu $L$. Damit ist aber $xR_Ly$ offensichtlich.

**3. → 1.:** Aus der Nerode-Relation konstruieren wir einen endlichen Automaten $A_{R_L} := (Z_{R_L}, \Sigma, \delta, Z_{\text{end}})$ der $L$ erkennt, wie folgt: Jeder der endlich vielen Äquivalenzklassen $[w]$ für $w \in \Sigma^*$ wird der Zustand $z_{[w]}$ zugeordnet, so dass die Mengen $Z_{R_L}$ und $\Sigma^*/R_L$ zueinander isomorph sind. Der Anfangszustand ist $z_{[\lambda]}$ und als Endzustände werden diejenigen Zustände $z_{[w]}$ gewählt, für die $w \in L$ ist. Diese Menge ist wohl definiert, denn aus der Definition von $R_L$ folgt, dass entweder alle Wörter einer Äquivalenzklasse $[w]_{R_L}$ in $L$ enthalten sind oder aber gar keines. Auch die Definition der Übergangsfunktion $\delta$ durch

$$\forall z_{[w]} \in Z_{R_L} : \forall x \in X : \delta(z_{[w]}, x) := z_{[wx]}$$

ist wohldefiniert, denn mit $[w_1] = [w_2]$ gilt $w_1R_Lw_2$ und wegen der Rechtsinvarianz von $R_L$ folgt auch $w_1xR_Lw_2x$ und somit $[w_1x] = [w_2x]$ für jedes $x \in X$. Die Folgezustände von $z_{[v]} \in \Sigma^*/R_L$ bei Eingabe von $w \in \Sigma^*$ ergeben sich als: $(z_{[v]})^w := z_{[vw]}$ und mit $(z_{[\lambda]})^w = z_{[w]} \in Z_{\text{end}}$ wird jedes Wort $w \in L$ von dem neuen Automaten akzeptiert. Als Tupel notiert schreibt sich der eben definierte endliche Automat als $A_{R_L} := (Z_{R_L}, X, \delta_{R_L}, z_{[\lambda]}, Z_{\text{end}})$ mit:

$Z_{R_L} := \{z_{[w]} \mid [w] \in \Sigma^*/R_L\}$,
$\delta_{R_L} : Z \times X \longrightarrow Z$ mit $\delta(z_{[w]}, x) := z_{[wx]}$ und

$Z_{\text{end}} := \{z_{[w]} \mid w \in L\}$.

Dieser endliche Automat ist ein initial zusammenhängender und vollständiger DFA!

□

Als Anwendung von Theorem 3.43 wir nun zeigen, dass eine einfache Sprache nicht regulär ist:

**3.44 Korollar**

*Die Sprache $DUP := \{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.*

**Beweis:** Betrachtet man einige (nicht alle, aber unendlich viele!) Äquivalenzklassen von $R_L$, so sieht man

$$\begin{aligned}
[ab] &= DUP = \{a^i b^i \mid i \in \mathbb{N}\}\\
[a^2b] &= \{a^{i+1}b^i \mid i \in \mathbb{N}\}\\
[a^3b] &= \{a^{i+2}b^i \mid i \in \mathbb{N}\}\\
&\vdots\\
[a^kb] &= \{a^{i+k-1}b^i \mid i \in \mathbb{N}\}\\
&\vdots
\end{aligned}$$

Man sieht, dass $[a^ib] \neq [a^jb]$ für $i \neq j$ gilt: $a^ib \not\equiv_{R_L} a^jb$, denn mit $w := b^{i-1}$ gilt $a^ibw \in DUP$ aber $a^jbw \notin DUP$. Somit sind also die unendlich vielen Klassen $[a^kb]$ der Nerode-Äquivalenz $R_L$ paarweise verschieden, und letztere daher nicht von endlichem Index. □

### 3.7.2 Minimale Automaten

Wie im wirklichen Leben, sind wir auch in der theoretischen Informatik daran interessiert, möglichst knappe Darstellungen zu verwenden. Zum Beispiel würde jeder Programmierer eine Sequenz von Anweisungen der Form „$X := X+1$“, „$X := X-1$“, „$X := X+1$“, „$X := X-1$“ einfach aus einem Programm streichen, da sie a m Ergebnis der Berechnung nichts ändert! Hier wollen wir insbesondere überflüssige Komponenten bei der Beschreibung von Sprachen vermeiden und suchen also nach möglichst kleinen Automaten für die zu beschreibende Sprache. Ein Beispiel für unterschiedlich große Automaten, welche dieselbe Sprache akzeptieren, war in Abbildung 3.14 auf Seite 73 gegeben.

**3.45 Definition**

*Zu jedem endlichen Automaten (NFA) $A = (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$ heiße die Menge $L(z) := \{w \in \Sigma^* \mid z \xrightarrow[w]{*} z', z' \in Z_{\text{end}}\}$ die **Leistung** des Zustandes $z$.*

*Zwei Zustände von $A$ heißen genau dann **äquivalent**, wenn $L(z) = L(z')$ ist. Wir notieren dies in der Form $z \equiv z'$. Nicht äquivalente Zustände, $z \not\equiv z'$, nennen wir **unterscheidbar**.*

*$A$ heißt genau dann **reduziert**, wenn keine zwei verschiedenen Zustände äquivalent sind.*

**3.46 Definition**

*Ein endlicher Automat heißt genau dann **minimal**, wenn er*

1. *vollständig,*
2. *initial zusammenhängend,*
3. *deterministisch und*
4. *reduziert ist.*

Diese Definition ist insofern merkwürdig, als bisher doch völlig unklar ist, wieso die Bezeichnung *minimal* gerechtfertigt ist. Wir werden also zunächst zeigen, dass kein vollständiger DFA, der zu einem minimalen Automaten äquivalent ist, weniger Zustände als dieser haben kann. Damit ist die Bezeichnung minimal gerechtfertigt. Sodann werden wir sehen, dass es (im Wesentlichen) nur *genau einen* minimalen DFA geben kann, der die gleiche Sprache akzeptiert.

**3.47 Theorem**
*Ein vollständiger DFA* $A = (Z, \Sigma, \delta, z_0, Z_{end})$ *ist genau dann minimal, wenn kein äquivalenter, vollständiger DFA weniger Zustände besitzt.*

**Beweis:** Es sind zwei Richtungen zu zeigen:

**„$\Rightarrow$“:** Wenn kein äquivalenter, vollständiger DFA weniger Zustände als $A$ besitzt, so muss $A$ entsprechend Definition 3.46 minimal sein.

**„$\Leftarrow$“:** Wenn $A$ minimal ist und $B = (Z', X, \delta', z'_0, Z'_{\text{end}})$ ein zu $A$ äquivalenter vollständiger DFA, dann gilt $|Z| \leq |Z'|$.

**zu „$\Rightarrow$“:** Wir zeigen die logisch äquivalente Kontraposition ($A$ nicht minimal $\rightarrow$ es gibt äquivalenten vollständigen DFA $B$ mit weniger Zuständen) der behaupteten Implikation. Wenn $A = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ *nicht* minimal ist, so kann das nach Definition 3.46 nur daran liegen, dass der vollständige DFA $A$ entweder nicht initial zusammenhängend ist (2. der Def.), oder äquivalente Zustände besitzt (4. der Def.). Falls $A$ nicht initial zusammenhängend ist, so gibt es mindestens einen Zustand, der von $z_0$ aus nicht erreicht wird, also entfernt werden kann ohne die akzeptierte Sprache zu verändern. Der neue Automat wäre ein äquivalenter vollständiger DFA mit weniger Zuständen. Falls $A$ nicht reduziert ist, so gibt es zwei verschiedene, aber äquivalente Zustände $p, q \in Z$. Identifiziert man diese Zustände miteinander, so erhält man wieder einen äquivalenten vollständigen DFA mit kleinerer Zustandsmenge.

**zu „$\Leftarrow$“:** Wir werden eine surjektive Abbildung $g : Z' \longrightarrow Z$ finden, womit dann alles gezeigt ist.

Es sei für jedes $z' \in Z'$ $g(z') := (z_0)^w$, falls in $B$ $z'_0 \xrightarrow[w]{*} z'$ für irgend ein Wort $w$ gilt. Wir zeigen:

**(i)** $g$ ist wohldefiniert, d.h tatsächlich eine Abbildung.

**(ii)** $g$ ist surjektiv.

**zu (i):** Seien $w, v \in \Sigma^*$ mit $w \neq v$ derart, dass in $B$ sowohl $z'_0 \xrightarrow[w]{*} z'$ als auch $z'_0 \xrightarrow[v]{*} z'$ gilt. Wir müssen einsehen, dass dann auch $(z_0)^w$ und $(z_0)^v$ in $A$ der gleiche Zustand ist:
Zunächst ist $L(z') = L((z_0)^w) = L((z_0)^v)$, denn für ein Wort $wu$ gilt $wu \in L(A)$ mit $L(A) = L(B) = L(z'_0)$ genau dann, wenn $u \in L((z_0)^w)$ gdw. $u \in L((z_0)^v)$ gdw. $u \in L(z')$ ist ($A$

und $B$ sind beide vollständige DFA's!). Somit sind also $(z_0)^w$ und $(z_0)^v$ im minimalen DFA $A$ äquivalente Zustände, müssen also identisch sein. Damit ist die Definition von $g$ mittels $g(z') := (z_0)^w$, falls $z'_0 \xrightarrow[w]{*} z'$ in $B$ möglich ist, unabhängig von der Wahl des Wortes $w$ und mithin eindeutig.

**zu (ii):** Es ist zu zeigen, dass es zu jedem Zustand $z \in Z$ ein $z' \in Z'$ gibt, so dass $g(z') = z$ ist. Da $A$ als minimaler Automat initial zusammenhängend ist, gibt es zu jedem $z \in Z$ wenigstens ein Wort $w \in \Sigma^*$ mit $z = (z_0)^w$. Da $B$ vollständig sein sollte, gibt es in $B$ für dieses $w$ dann immer auch einen Zustand $z' \in Z'$ mit $z' = (z'_0)^w$ und also gilt nach Definition gerade $g(z') = (z_0)^w = z$. Dies liefert die verlangte Surjektivität von $g$.

□

Als nächstes wollen wir zeigen, dass minimale DFA *eindeutig* bestimmt sind, d.h. je zwei äquivalente minimale DFA sind modulo Namensgebung der Zustände identisch.

**3.48 Theorem**

*Je zwei äquivalente, minimale DFA sind isomorph, d.h., sie besitzen bis auf die Bezeichnungen der Zustände das gleiche Zustandsdiagramm.*

**Beweis:** Seien $A_1$ und $A_2$ mit $A_i := (Z_i, X, \delta_i, z_{0,i}, Z_{end,i}), i = 1, 2$ zwei minimale Automaten die die gleiche Sprache $L := L(A_1) = L(A_2)$ akzeptieren. Da ein minimaler endlicher Automat gemäß seiner Definition 3.46 ein vollständiger und deterministischer endlicher Automat ist, muss nach Theorem 3.47 zunächst $|Z_1| = |Z_2|$ gelten. Um nun eine Bijektion $f : A_1 \longrightarrow A_2$ zwischen den Zustandsgraphen herzustellen, definieren wir $f(z_{0,1}) := z_{0,2}$, $Z_{end,2} := \{f(z) \mid z \in Z_{end,1}\}$ und $\forall x \in X, z \in Z_1 : f(\delta_1(z, x)) := \delta_2(f(z), x)$. Die Abbildung $f$ bildet also jede Kante des Zustandsgraphen von $A_1$ auf eine Kante des Zustandsgraphen von $A_2$ ab. Tatsächlich ist $f$ ein Isomorphismus zwischen den Kantenmengen. Wenn $(z_{0,1}, w_1, z_{1,1}), (z_{1,1}, w_2, z_{2,1}), \ldots, (z_{n-1,1}, w_n, z_{n,1})$, mit $(z_{i-1,1}, w_i, z_{i,1})$ eine Erfolgsrechnung in $A_1$ ist, so ist auch $(f(z_{0,1}), w_1, f(z_{1,1})), \ldots, (f(z_{n-1,1}), w_n, f(z_{n,1}))$ eine Erfolgsrechnung in $A_2$ und umgekehrt. □

Der minimale endliche Automat ist weiterhin (bis auf Isomorphie) identisch mit dem initial zusammenhängenden, vollständigen DFA $A_{R_L}$ der im Beweis zu Theorem 3.42, im Schritt **3.** → **1.**, konstruiert wurde.

**3.49 Korollar**

*Der im Beweis zu Theorem 3.43 konstruierte endliche Automat $A_{R_L}$ ist ein minimaler DFA.*

**Beweis:** Sei $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein beliebiger vollständiger DFA mit $L := L(A)$. Nach Theorem 3.43, Beweisschritt 1. $\longrightarrow$ 2., existiert eine rechtsinvariante Äquivalenzrelation $R_A$ und $L$ ist endliche Vereinigung jener Äquivalenzklassen **mod** $R_A$ die zu den Endzuständen von $A$ gehören. Damit folgt $\text{ind}(R_A) \leq |Z|$. Nach dem Beweis von 2. $\longrightarrow$ 3. folgt, dass $R_L$ eine Vergröberung von $R_A$ ist, mithin $\text{ind}(R_L) \leq \text{ind}(R_A)$ gilt. Aus dem Beweisschritt 3. $\longrightarrow$ 1. folgt, dass $\text{ind}(R_L) = |Z_{R_L}|$ und also $|Z_{R_L}| \leq |Z|$ ist. □

Wir können nun mit Fug und Recht von *dem* minimalen endlichen Automaten sprechen, wohl wissend, dass nach den Theoremen 3.29 und 3.50 *nicht*deterministische endliche Automaten logarithmisch viel kleiner sein können! Dies zeigen wir im Beweis zu Theorem 3.50

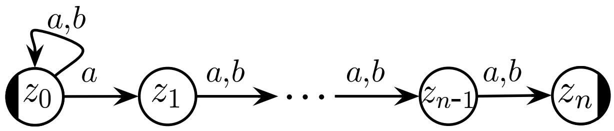



Abbildung 3.15: Ein NFA für die Sprache $L_n := \{w \in \{a,b\}^* \mid \text{das } n\text{-letzte Symbol in } w \text{ ist } a\}$

**3.50 Theorem**

*(i) Es gibt einen $\lambda$-freien, buchstabierenden NFA mit $n+1$ Zuständen, der die Sprache $L_n := \{w \in \{a,b\}^* \mid$ das $n$-letzte Symbol in $w$ ist ein $a\}$ akzeptiert.*

*(ii) Jeder DFA, der die Menge $L_n$ akzeptiert hat $2^n$ Zustände.*

**Beweis:**

zu (i): Der NFA aus Abbildung 3.15 akzeptiert die Sprache $L_n$.

**zu (ii):** Sei $B = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein DFA mit $L(B) = L_n$ und seien $w_1, w_2 \in \{a,b\}^n$ verschiedene Wörter. Wir zeigen $(z_0)^{w_1} \neq (z_0)^{w_2}$: Da $w_1$ und $w_2$ verschieden sind, gibt es mindestens einen Buchstaben, an dem sich die beiden Wörter unterscheiden, d.h., es gibt Wörter $u_1, u_2, v \in \{a,b\}^*$ mit o.B.d.A. $w_1 = u_1 a v$ und $w_2 = u_2 b v$. Für $v' \in \{a,b\}^*$ mit $|v'| := n - |v| - 1$ gilt nun 1.): $w_1 v' \in L_n$ und 2.): $w_2 v' \notin L_n$, denn der $n$-letzte Buchstabe von $w_1 v' = u_1 a v v'$ ist ein $a$ aber der von $u_2 b v v'$ nicht. Wäre nun aber $(z_0)^{w_1} = (z_0)^{w_2} = z'$, so hieße dies doch $(z')^{v'} \in Z_{\text{end}}$ wegen 1.) und gleichzeitig auch $(z')^{v'} \notin Z_{\text{end}}$ wegen 2.). Dies ist aber ein Widerspruch, folglich kann die Annahme $(z_0)^{w_1} = (z_0)^{w_2}$ nicht richtig gewesen sein. Da es $2^n$ paarweise verschiedene Wörter der Länge $n$ in $\{a,b\}^*$ gibt, folgt die Existenz von ebenso vielen verschiedenen Zuständen im Automaten $B$.

□

Auf die gleiche Weise können wir nun auch Theorem 3.29 beweisen.

**Beweis (zu Theorem 3.29):** Sei $K_n := L(A_n)$ die vom NFA $A_n$ akzeptierte Sprache. Es ist leicht zu sehen, dass neben dem leeren Wort $\lambda$ kein anderes Wort der Länge kleiner als $n$ akzeptiert wird und alle nicht leeren Wörter $w \in K_n$ die Form $w = uav$ mit $v \in \{a,b\}^{n-1}$ besitzen.

□

Ein minimaler DFA besitzt keine äquivalenten Zustände. Daher reicht es aus, in einem gegebenen initial zusammenhängenden vollständigen deterministischen endlichen Automaten äquivalente Zustände zu identifizieren, um einen äquivalenten und minimalen DFA mit einer neuen Zustandsmenge zu konstruieren. Jeder Zustand in diesem neuen Automaten ist die Äquivalenzklasse der Zustände gleicher Leistung des alten Automaten.

**3.51 Definition**

*Sei $A = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein DFA, und für jeden Zustand $z \in Z$ seine Äquivalenzklasse $[z] := \{z' \mid z' \equiv z\}$ die Menge aller Zustände mit gleicher Leistung (vgl. Def. 3.41). Der* ***Äquivalenzautomat*** *$A'$ zu $A$*

*ist wie folgt erklärt:* $A' := (Z', X, \delta', [z_0], Z'_{end})$, *wobei* $Z' := \{[z] \mid z \in Z\}$, $Z'_{end} := \{[z] \mid z \in Z_{end}\}$ *und* $\delta'([z_1]_A, x) = [z_2]_A$ *genau dann, wenn* $\delta(p, x) = q$ *für irgendwelche Zustände* $p \in [z_1]$ *und* $q \in [z_2]$ *gilt.*

**3.52 Theorem**

*Der Äquivalenzautomat* $A'$ *zu einem initial zusammenhängenden vollständigen DFA* $A$ *ist minimal und akzeptiert die gleiche Sprache.*

**Beweis:** Aus Definition 3.41 folgt sofort, dass die Endzustandsmenge wohldefiniert ist, denn es impliziert $z \equiv z'$ entweder $z, z' \in Z_{\text{end}}$ oder $z, z' \notin Z_{\text{end}}$. Die neue Übergangsfunktion $\delta'$ ist wohldefiniert, weil aus $z \equiv z'$ für jedes $a \in X$ stets $(z)^a \equiv (z')^a$ folgt. Unter Rückgriff auf Definition 3.41 sieht man dies leicht:

(1) $z \equiv z'$ impliziert $L(z) = L(z')$ oder

(2) $\forall w \in \Sigma^* :\ (z)^w \in Z_{\text{end}}$ genau dann, wenn $(z')^w \in Z_{\text{end}}$. Also auch

(3) $\forall a \in X, \forall w \in \Sigma^* :\ (z)^{aw} \in Z_{\text{end}}$ genau dann, wenn $(z')^{aw} \in Z_{\text{end}}$ und es folgt

(4) $(z)^a \equiv (z')^a$.

Es ist noch zu zeigen, dass $A$ und $A'$ die gleiche Sprache akzeptieren:

Sei dazu $w \in \Sigma^*$ und $z_1, z_2, \ldots, z_n$ die von $A$ beim Lesen von $w$ durchlaufene Zustandsfolge. Nach der Definition von $A'$, speziell $K'$, durchläuft $A'$ die Zustandsfolge $[z_1], [z_2], \ldots, [z_n]$. $A$ akzeptiert $w$, genau dann, wenn $z_n \in Z_{\text{end}}$ ist. Der Äquivalenzklassenautomat akzeptiert $w$ genau dann, wenn $[z_n] \in Z'_{\text{end}}$ ist. Entsprechend der Definition gilt aber $z_n \in Z_{\text{end}}$ genau dann, wenn $[z_n] \in Z'_{\text{end}}$ ist.

Schließlich muss gezeigt werden, dass der Äquivalenzautomat minimal ist:

Dies folgt aus Korollar 3.49, wenn wir zeigen, dass der Äquivalenzautomat höchstens $\text{ind}(R_L)$ viele Zustände besitzt. Dies folgt nun aber schon, wenn im Äquivalenzautomaten zwei Wörter $w_1$ und $w_2$ mit $w_1 R_L w_2$ stets zum gleichen Zustand $[(z_0)^{w_1}] = [(z_0)^{w_2}]$ führen, oder formaler: $\forall w_1, w_2 \in \Sigma^* : (w_1 R_L w_2) \to ((z_0)^{w_1} \equiv (z_0)^{w_2})$.

$$\begin{array}{lll} w_1 R_L w_2 & \to & \forall v \in \Sigma^* : w_1 v \in L \leftrightarrow w_2 v \in L \\ & \to & \forall v \in \Sigma^* : (z_0)^{w_1 v} \in Z_{\text{end}} \leftrightarrow (z_0)^{w_2 v} \in Z_{\text{end}} \\ & \to & \forall v \in \Sigma^* : ((z_0)^{w_1})^v \in Z_{\text{end}} \leftrightarrow ((z_0)^{w_2})^v \in Z_{\text{end}} \\ & \to & (z_0)^{w_1} \equiv (z_0)^{w_2} \\ & \to & [(z_0)^{w_1}] = [(z_0)^{w_2}] \end{array}$$

□

Bei Kenntnis der Äquivalenklassen bezüglich $\equiv$ kann der Äquivalenzklassenautomat in einer Zeit proportional zum Produkt $|Z| \cdot |X|$ konstruiert werden. Schwieriger scheint die Bestimmung der Relation $\equiv$ zu sein, denn gemäß der Definition müssen immer alle Zustände und die unendliche Menge aller Wörter in $\Sigma^*$ betrachtet werden.

Man kann den minimalen DFA zu jedem gegebenden DFA mit $n$ Zuständen in $c \cdot n \cdot \log n$ Schritten konstruieren. Die hier dargestellte einfachere Methode (siehe z.B. [Wegener] oder [Hopcroft&Ullman]) arbeitet mit etwas schlechteren $d \cdot n^2$ Schritten. Die jeweiligen Konstanten $c$ und $d$ hängen dabei stets auch von der Größe des verwendeten Eingabealphabets $\Sigma$ ab. Man macht sich bei diesem Verfahren zu Nutze, dass es leichter ist, die Nichtäquivalenz zweier Zustände $z$ und $z'$ zu bezeugen, denn dazu benötigt man nur ein sogenanntes $z$ **und** $z'$ **unterscheidendes Wort** $w \in \Sigma^*$ mit $(z)^w \in Z_{\text{end}}$ und $(z')^w \notin Z_{\text{end}}$ oder umgekehrt, mit $(z)^w \notin Z_{\text{end}}$ und $(z')^w \in Z_{\text{end}}$.

**3.53 Algorithmus (Bestimmung äquivalenter Zustände eines vDFA)**
**Eingabe:** $A = (Z, \Sigma, K, z_0, Z_{\text{end}})$ ein DFA mit $Z = \{z_1, z_2, \ldots, z_n\}$.

**Ausgabe:** Markierung aller Zustandspaarmengen $\{z_i, z_j\}$ mit $z_i \not\equiv z_j$ und $i < j$.

Ein gerichteter Graph $G_A := (V, E)$ mit $V := \{v \mid v \in 2^Z \text{ und } |v| = 2\}$ wird als Datenstruktur verwendet. Die Knoten in $V$ sind also Paarmengen $\{z_i, z_j\}$ von unterschiedlichen Zuständen und können gefärbt werden. Zu Beginn ist kein Knoten von $G_A$ gefärbt. Im weiteren Verlauf werden gerichtete Kanten entgegen der Richtung der Kanten im DFA gezogen oder entfernt und ungefärbte Knoten weiß oder schwarz gefärbt. Die geschwärzten Knoten geben dann jeweils Paare von inäquivalenten Zuständen an.

**Initialisierung:** Jeder Knoten $\{z_i, z_j\}$, für den entweder ($z_i \in Z_{\text{end}}$ und $z_j \notin Z_{\text{end}}$) oder aber ($z_i \notin Z_{\text{end}}$ und $z_j \in Z_{\text{end}}$) ist, wird schwarz gefärbt. D.h., das die Zustände $z_i$ und $z_j$ unterscheidende Wort ist $\lambda$.

Falls sich im weiteren Verlauf der Konstruktion irgendwann $z_i$ und $z_j$ als inäquivalent erweisen, so können alle Knoten, die von dem geschwärzten Knoten $\{z_i, z_j\}$ aus über noch zu definierende Kanten erreichbar sind, ebenfalls als inäquivalent betrachtet und daher auch schwarz gefärbt werden.

**Färbekonstruktion:** Solange noch ungefärbte Knoten in dem Graphen existieren, tue man folgendes:

**Schritt 1:** Wähle einen ungefärbten Knoten $\{p, q\}$ und färbe diesen weiß.

**Schritt 2:** Für jedes $a \in \Sigma$, für das $(p)^a \neq (q)^a$ ist, überprüfe, ob der Knoten $\{(p)^a, (q)^a\}$ schwarz markiert ist.

**wenn ja,** so wird auch $\{p, q\}$ schwarz markiert und alle von diesem Knoten über Kanten erreichbaren noch nicht schwarzen Knoten ebenfalls. Danach werden alle Kanten, die im Knoten $\{p, q\}$ enden, entfernt.

**wenn nein,** so zeichne eine Kante $\{(p)^a, (q)^a\} \longrightarrow \{p, q\}$.

**Ergebnis:** Wenn der Knoten $\{p, q\}$ am Ende weiß ist, so gilt $p \equiv q$.
Wenn der Knoten $\{p, q\}$ am Ende schwarz ist, so gilt $p \not\equiv q$

**Beweis der Korrektheit von Algorithmus 3.53:**

1. Zunächst beweisen wir die Termination des Algorithmus:
   Die „Solange“-Schleife terminiert, denn jeder Knoten wird höchstens einmal weiß und höchstens einmal schwarz gefärbt.

2. Zum Beweis der Korrektheit zeigen wir:

   **(i)** Falls $\{p, q\}$ schwarz ist, dann folgt $p \not\equiv q$, und

   **(ii)** falls $p \not\equiv q$ ist, so ist der Knoten $\{p, q\}$ am Ende schwarz gefärbt.

Es ist festzuhalten, dass aus der Inäquivalenz zweier Zustände $p \not\equiv q$ stets die Existenz eines $p$ und $q$ unterscheidenden Wortes, eines sogenannten Zeugen (*witness*) $w \in \Sigma^*$ folgt.

**zu (i):** Wir führen eine Induktion über die Zahl $d$ der Durchläufe der „Solange"-Schleife (Schritte 1 und 2) durch.

**Induktions-Basis:** Für $d = 0$ zeigt die Initialisierung, dass $\{p, q\}$ geschwärzt ist, weil $\lambda$ das unterscheidende Wort ist.

**Induktions-Annahme:** Sei **(i)** richtig für alle in den ersten Schleifendurchläufen geschwärzten Knoten.

**Induktions-Schritt:** Im nächsten Durchlauf der Schritte 1 und 2 wird $\{p, q\}$ geschwärzt, falls einer der folgenden Fälle zutrifft:

**direkte Färbung:** $\exists x \in \Sigma : \{(p)^x, (q)^x\}$ war schon schwarz, d.h., es gibt einen Zeugen $w \in \Sigma^*$, der $(p)^x$ und $(q)^x$ unterscheidet. Mithin unterscheidet das Wort $xw$ die beiden Zustände $p$ und $q$ und es gilt $p \not\equiv q$.

**indirekte Färbung:** Ein anderer Knoten $\{s, t\}$ wird direkt geschwärzt und der Knoten $\{p, q\}$ ist auf einem Pfad von $\{s, t\}$ erreichbar, so dass daher $\{p, q\}$ geschwärzt wird. Nach der Definition des Ziehens neuer Kanten in Schritt 2 gibt es ein Wort $v \in \Sigma^*$ mit $s = (p)^v$ und $t = (q)^v$. Da nun aber $\{s, t\}$ nur dann geschwärzt wurde, wenn es ein $s$ und $t$ unterscheidendes Wort $w$ gab, gilt für dieses $w$ dann, dass genau einer der Zustände $(s)^w$ und $(t)^w$ ein Endzustand ist. Folglich ist auch genau einer der Zustände $((p)^v)^w$ und $((q)^v)^w$ ein Endzustand. Das $p$ und $q$ unterscheidende Wort ist also $vw$.

Damit ist **(i)** durch Induktion gezeigt.

**zu (ii):** Für jedes $p \not\equiv q$ gibt es ein kürzestes Wort $w$, welches $p$ und $q$ unterscheidet. Wir wählen Induktion über die Länge $k$ dieser Wörter.

**Induktions-Basis:** Gilt $|w| = 0$, so muss $\lambda$ das unterscheidende Wort sein. Da der endliche Automat ein DFA war, kann nur genau einer der beiden Zustände ein Endzustand sein und in diesem Fall wurde der Knoten $\{p, q\}$ bei der Initialisierung geschwärzt. Einmal geschwärzte Knoten werden aber auch nicht wieder anders gefärbt.

**Induktionsannahme:** Alle Knoten $\{s, t\}$, bei denen $s$ und $t$ durch Wörter mit Längen bis zu $k$ unterscheidbar waren, wurden irgendwann geschwärzt.

**Induktionsschritt:** Sei $p \not\equiv q$ unterscheidbar mit dem kürzestem Wort $v = aw$ mit $|v| = k + 1$ und $a \in X$, $w \in \Sigma^*$. Dann unterscheidet $w$ die Zustände $(p)^a$ und $(q)^a$ und der Knoten $\{(p)^a, (q)^a\}$ wird wegen der Induktionsannahme irgendwann geschwärzt werden. Irgendwann wird auch $\{p, q\}$ unter den noch nicht gefärbten Knoten ausgewählt. Wenn dann für irgend ein $x \in \Sigma$ ($x = a$ ist dabei möglich) der Knoten $\{(p)^x, (q)^x\}$ schon schwarz ist, so wird in direkter Färbung auch sogleich $\{p, q\}$ geschwärzt. Falls jedoch $\{(p)^x, (q)^x\}$ noch für kein $x \in \Sigma$ schwarz ist, dann werden die Kanten $\{(p)^x, (q)^x\} \longrightarrow \{p, q\}$ für jedes $x \in \Sigma$ gezeichnet. Da nach der Induktionsannahme irgendwann der Knoten $\{(p)^a, (q)^a\}$ geschwärzt wird, wird zu diesem Zeitpunkt auch $\{p, q\}$ in indirekter Färbung geschwärzt.

Damit ist die Korrektheit des Färbealgorithmus gezeigt. □

**3.54 Theorem**

*Der Färbealgorithmus zum Auffinden äquivalenter Zustände arbeitet in $c \cdot |\Sigma| \cdot |Z|^2$ Schritten, wobei $c$ eine Konstante ist.*

**Beweisskizze:** Jeder Knoten wird in Schritt 1 maximal einmal betrachtet. Jedesmal werden in Schritt 2 dazu $|\Sigma|$ Knoten $\{(p)^x, (q)^x\}$ untersucht. Für jeden dieser Knoten kann $\{p, q\}$ und alle von hier aus erreichbaren Knoten, das sind maximal $|Z|$ viele, besucht werden um diese zu schwärzen oder um Kanten zu ziehen. Insgesamt ergibt sich die Zahl von höchstens $c \cdot |X| \cdot |Z|^2$ Besuchen von einzelnen Knoten. Genauere Beweise finden sich in der Literatur. $\square$

Auf Grund diese Ergebnisses ist der minimale DFA in höchstens quadratischer Zeit aus einem DFA konstruierbar.

Bei besserer Programmierung kann jedoch ein Algorithmus formuliert werden, der das Finden äquivalenter Zustände sogar in nur $c \cdot |Z| \cdot log(|Z|)$ vielen Schritten ermöglicht. Wir wollen an dieser Stelle noch nicht auf diese Methode des *Dynamischen Programmierens* eingehen. Dazu verweisen wir auf andere Veranstaltungen, zum Beispiel auch auf die Vorlesung F3 („Berechenbarkeit und Komplexität“) in diesem Zyklus.

## 3.8 Das Pumping-Lemma

Die Frage, ob alle Mengen regulär sind, muss sofort verneint werden! Eine spezielle Menge, die Sprache $DUP$ hatten wir in Korollar 3.44 bereits kennengelernt und als nicht-regulär nachgewiesen.

Um von einer beliebig vorgegebenen Menge von Wörtern nachzuweisen, dass es keinen endlichen Automaten geben kann, der sie akzeptiert, gibt es jedoch noch weitere, einfacher anzuwendende Methoden. Als wichtiges Hilfsmittel dient hier das sogenannte „uvw-Theorem“ (*pumping lemma*).

**3.55 Theorem (Pumping-Lemma der regulären Sprachen, $uvw$-Theorem)**
*Zu jeder regulären Menge $R \in \mathcal{REG}$ gibt es eine Zahl $n \in I\!N$, so dass jedes Wort $z \in R$ mit $|z| \geq n$ zerlegt werden kann in die Form $z = uvw$ mit:*

*(i)* $|uv| \leq n$

*(ii)* $|v| \geq 1$ *und*

*(iii)* $\{u\}\{v\}^*\{w\} \subseteq R$

**Beweis:** Sei $A = (Z, \Sigma, K, z_0, Z_{\text{end}})$ ein DFA der $R$ akzeptiert, d.h.: $L(A) = R$. Wähle $n := |Z|$. Sei nun $z = x_1x_2 \ldots x_m \in R$, mit $m \geq n$ und $x_i \in \Sigma$, ein beliebiges Wort der Sprache. Dann sind doch die $m + 1$ Zustände $z_0, (z_0)^{x_1}, (z_0)^{x_1x_2}, \ldots, (z_0)^{x_1x_2\ldots x_m}$ alle von $z_0$ aus erreichbar und werden auf diesem Erfolgspfad für die Akzeptierung von $z$ besucht. Da aber nur maximal $n \leq m$ viele davon verschieden sein können, muss nach dem Schubfachprinzip (vergl. 2.19) spätestens mit dem $(n+1)$-ten, ein Zustand in dieser Folge zweimal vorgekommen sein.

Sei also für $0 \leq i < j \leq n$ gerade $(z_0)^{x_1x_2\ldots x_i} = (z_0)^{x_1x_2\ldots x_j}$, dann finden wir mit $u := x_1x_2 \ldots x_i$, $v := x_{i+1}x_{i+2} \ldots x_j$ und $w := x_{j+1}x_{j+2} \ldots x_m$ gerade die verlangte Zerlegung von $z$, die (i) und (ii) erfüllt. Dass auch (iii) gilt, sieht man leicht ein, denn mit $z' := (z_0)^u$ gilt doch $z' = (z')^\lambda = (z')^v = (z')^{vv} = \ldots$. Also werden alle Wörter der Form $u \cdot v^i \cdot w$, für jedes $i \geq 0$, von $A$ auch akzeptiert. $\square$

Da jede reguläre Menge auch von einem minimalen DFA akzeptiert wird, kann für die Zahl $n$ im $uvw$-Theorem die Anzahl der Zustände des minimalen Automaten, der $R \in \mathcal{R}eg$ akzeptiert, angenommen werden.

Obwohl die Folgerung des $uvw$-Theorem (d.h. die Aufteilbarkeit der „langen" Wörter mit den angegebenen Eigenschaften) für alle regulären Mengen gilt, gibt es auch nicht-reguläre Mengen, für die diese Aussagen gelten. Daher kann mit Hilfe des $uvw$-Theorems nicht bewiesen werden, dass eine Menge tatsächlich regulär ist, sondern nur umgekehrt, dass sie *nicht* regulär sein kann!

**3.56 Beispiel (Anwendung des $uvw$-Theorems)**
Wir wollen mit Hilfe des uvw-Theorems zeigen, dass die Sprache $DUP := \{a^n b^n \mid n \in I\!N\}$ nicht regulär ist: Wäre $DUP$ regulär, so gäbe es eine Zahl $k \in I\!N$, für die die Eigenschaften des Pumping-Lemmas zuträfen (genauer: für die die Konklusion des Pumping Lemmas zuträfe). Sei also $z := a^k b^k \in DUP$ eines der Wörter mit $|z| > k$. Jede Zerlegung von $z$ als $z = uvw$ mit $|uv| \leq k$ bedeutet $v = a^m$ für ein $m \in I\!N$ mit $1 \leq m \leq k$. Damit müßte aber auch das Wort $a^{k-m}b^k$ ein Element der Sprache $DUP$ sein, was nicht stimmt. Da alle Schritte korrekt aus der Annahme abgeleitet wurden, und dann ein Widerspruch entstand, kann nur die Annahme falsch gewesen sein. Damit ist $DUP \notin \mathcal{REG}$ mit einer weiteren Methode nachgewiesen.

Die im $uvw$-Theorem gefolgerten Eigenschaften treffen zum Beispiel auf die Sprache $L := \{c^k a^l b^m \mid k, l, m \in I\!N : k = 0 \text{ oder } l = m\}$ zu. Der Versuch, mit dem $uvw$-Theorem zu beweisen, dass $L$ nicht regulär ist muss deshalb fehlschlagen. Wenn nämlich $z \in L$ den Buchstaben „$c$" nicht enthält, so gilt dies auch für alle Wörter $uv^iw$, ganz gleich welche Zerlegung gewählt wurde.

Enthält das Wort $z$ das Symbol „$c$", so wählen wir eine Zerlegung mit $u = \lambda$, $v = c$ und $w$ als geeignetem Suffix. Alle Wörter der Form $uv^iw$ sind dann wieder Elemente der Menge $L$. Also kommt kein Widerspruch zu der Annahme $L$ sei regulär zustande, obwohl $L$ eben nicht regulär ist, wie wir später auf Seite 89 zeigen werden.

Alle bisher definierten Klassen von Sprachen waren in Sinne von Definition 3.5 Sprachfamilien. Operationen, die man auf Sprachen anwendet, wie z.B. den mengentheoretischen Durchschnitt kann man zwar auch auf die Sprachfamilien anwenden, da diese ja selbst Mengen sind, jedoch ist dies im allgemeinen nicht das Ziel. Offensichtlich ergeben sich dabei wenig neue Gesichtspunkte: $\mathcal{R}at \cap \mathcal{R}eg = \mathcal{R}at$ ist bekannt, denn jede rationale Menge ist auch eine reguläre Menge.

## 3.9 Operatoren auf Sprachfamilien

Was wir eigentlich betrachten wollen, ist hier eher der Durchschnitt der Sprachen aus den beiden Familien. Man bezeichnet in der AFL-Theorie (*theory of abstract formal languages*) dann auch korrekterweise diese Operationen auf den Sprachen als **Operatoren** auf den Sprachfamilien, um so den Unterschied deutlich zu machen. Man einigte sich auf folgende Notation:

**3.57 Definition**
*Für Sprachfamilien $\mathcal{L}_1$ und $\mathcal{L}_2$ sei*

$$\mathcal{L}_1 \vee \mathcal{L}_2 \quad := \quad \{L \mid \exists L_1 \in \mathcal{L}_1, \exists L_2 \in \mathcal{L}_2 : L = L_1 \cup L_2\}$$

$$\begin{aligned}\mathcal{L}_1 \wedge \mathcal{L}_2 &:= \{L \mid \exists L_1 \in \mathcal{L}_1, \exists L_2 \in \mathcal{L}_2 : L = L_1 \cap L_2\}\\ \mathcal{L}_1^* &:= \{L^* \mid L \in \mathcal{L}_1\}\end{aligned}$$

Obschon damals noch nicht so notiert, zeigten wir $\mathcal{R}eg \vee \mathcal{R}eg = \mathcal{R}eg$ in Theorem 3.31 und offensichtlich gilt nach Theorem 3.33 auch $\mathcal{R}eg^* := \{L^* \mid L \in \mathcal{R}eg\} = \mathcal{R}eg$, wobei wir den Operator $*$ genauso bezeichnen können wie den Stern-Abschluss (Kleene'sche-Hülle) einer formalen Sprache. Letzteres Vorgehen nennt man auch **kanonische Erweiterung** einer Operation.

Neben diesen offensichtlichen Eigenschaften regulärer Mengen sind jedoch auch besonders die mengentheoretische Komplementbildung (bzw. der Differenzbildung) und der Durchschnitt bedeutsam.

**3.58 Theorem**

*Mit $L \in \mathcal{A}kz(\Sigma)$ ist auch $\overline{L} := \Sigma^* \setminus L \in \mathcal{A}kz(\Sigma)$.*

**Beweis:** Sei $A := (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein vollständiger DFA mit $L = L(A)$. Den vDFA $C_{\overline{A}}$ mit $L(C_{\overline{A}}) = \overline{L}$ erhält man sofort als $C_{\overline{A}} := (Z, \Sigma, \delta, z_0, Z \setminus Z_{\text{end}})$. Da $A$ vollständig war, ist nun jede der nicht akzeptierenden Rechnungen von $A$ eine Erfolgsrechnung in $C_{\overline{A}}$ und jede Erfolgsrechnung von $A$ ist in $C_{\overline{A}}$ nicht mehr akzeptierend. □

Da nach den Gesetzen von de Morgan $A \cap B = \overline{\overline{A} \cup \overline{B}}$ gilt, wissen wir nach dem Vorangegangenen (siehe: Theorem 3.31) auch, dass der Durchschnitt zweier regulärer Mengen wieder regulär sein muss. Die Hintereinanderausführung der bisher kennengelernten Umformungen der endlichen Automaten wäre allerdings zur Konstruktion eines Automaten $C_{A\cap B}$ mit $L(C_{A\cap B}) = L(A) \cap L(B)$ sehr zeitaufwendig und unbefriedigend. Wir wollen daher die wichtige Konstruktion des Produktautomaten vorstellen.

**3.59 Theorem**

*Mit $L_1 \in \mathcal{A}kz(\Sigma_1)$ und $L_2 \in \mathcal{A}kz(\Sigma_2)$ ist auch $L_1 \cap L_2 \in \mathcal{A}kz(\Sigma_1 \cap \Sigma_2)$.*

**Beweis (Konstruktion eines vDFA für $L_1 \cap L_2$):** Seien $A := (Z_1, \Sigma_1, \delta_1, z_{1,0}, Z_{1,\text{end}})$ und $B := (Z_2, \Sigma_2, \delta_2, z_{2,0}, Z_{2,\text{end}})$ vDFA's mit $L_1 = L(A)$ und $L_2 = L(B)$. Der vDFA $C_{A\cap B} := (Z_3, \Sigma_3, \delta_3, z_{3,0}, Z_{3,\text{end}})$ mit $L(C_{A\cap B}) = L_1 \cap L_2$ wird **Produktautomat** genannt, weil dessen Zustandsmenge das kartesische Produkt der Zustandsmengen $Z_1$ und $Z_2$ ist. Seine Bestandteile sind folgendermaßen spezifiziert:

$$\begin{aligned}Z_3 &:= Z_1 \times Z_2,\\ \Sigma_3 &:= \Sigma_1 \cap \Sigma_2,\\ \delta_3((z_1, z_2), x) &:= (\delta_1(z_1, x), \delta_2(z_2, x)) \text{ für alle Zustände} (z_1, z_2) \in Z_3,\\ z_{3,0} &:= (z_{1,0}, z_{2,0}),\\ Z_{3,\text{end}} &:= Z_{1,\text{end}} \times Z_{2,\text{end}}.\end{aligned}$$

Es ist offensichtlich, dass zu jeder Erfolgsrechnung für ein Wort $w \in L_1 \cap L_2$ Erfolgsrechnungen in $A$ und in $B$ existieren, für die in der ersten und in der zweiten Komponente die entsprechende Erfolgsrechnung in $C_{A\cup B}$ existiert. Umgekehrt, entspricht einer Erfolgsrechnung in $C_{A\cup B}$, die im Zustand $(z_1, z_2)$ mit $z_1 \in Z_{1,\text{end}}$ und $z_2 \in Z_{2,\text{end}}$ endet, sowohl eine Erfolgsrechnung in $A$ als auch eine ebensolche in $B$. □

**3.60 Definition**

*Eine Sprachfamilie $\mathcal{L}$ ist **abgeschlossen unter einem Operatior** $o : 2^X \times 2^X \longrightarrow 2^X$ genau dann, wenn $o(L_1, L_2) \in \mathcal{L}$, für alle $L_1, L_2 \in \mathcal{L}$ gilt.*

*Wir nennen in diesem Fall den Operator o einen* ***Abschlußoperator*** *der Familie $\mathcal{L}$ und sagen, dass die Familie $\mathcal{L}$ gegenüber o abgeschlossen ist.*

Wir hatten bisher gesehen, dass die Familie $\mathcal{R}eg$ der regulären Mengen abgeschlossen ist gegenüber den Operatoren: Vereinigung ($\vee$), (Komplex-)Produkt, Sternbildung, Komplement, Mengendifferenz und Durchschnitt ($\wedge$). Das Wechseln des Alphabets durch Austausch einzelner Symbole ist offensichtlich auch keine Schwierigkeit. Man ersetze die entsprechenden Symbole in dem die Menge beschreibenden rationalen Ausdruck. Diese Idee wird nun mehrfach verallgemeinert, indem für einzelne Symbole nun ganze Sprachen *substituiert* werden:

**3.61 Definition**

*Eine* ***Substitution*** *ist ein Homomorphismus $s : \Sigma^* \longrightarrow 2^{Y^*}$, wobei $X$ und $Y$ Alphabete sind und $s$ folgende Eigenschaften besitzt:*

1. *Für jedes $a \in X$ ist $s(a) \subseteq Y^*$ definiert.*
2. $s(\lambda) := \{\lambda\}$.
3. $\forall u, v \in \Sigma^* : s(uv) = s(u) \cdot s(v)$

*Ist $s(a)$ regulär (bzw. endlich) für jedes $a \in X$, dann heißt $s$ eine* ***reguläre*** *bzw. eine* ***endliche*** *Substitution.*

*Für eine Sprache $L \subseteq \Sigma^*$ werde $s$ kanonisch (in natürlicher Weise) erweitert durch:*

$$s(L) := \bigcup_{w \in L} s(w)$$

**3.62 Beispiel**

Für $s : \{0,1\}^* \longrightarrow 2^{\{a,b\}^*}$ mit $s(0) := \{a\}, s(1) := \{b\}^*$ ergibt sich $s(0^*(0+1)1^*) = a^*(a+b^*)(b^*)^* = a^*b^*$, wobei wir die Mengen durch nur leicht vereinfachte, rationale Ausdrücke beschrieben haben.

**3.63 Theorem**

*Die Familie der regulären Mengen ist gegenüber regulären Substitutionen abgeschlossen.*

**Beweis:** Jede reguläre Menge $R$ wird durch einen rationalen Ausdruck dargestellt. Ersetzt man nun jedes Symbol $a$ in diesem Ausdruck durch den rationalen Ausdruck der $s(a)$ beschreibt, so ergibt sich ein rationaler Ausdruck für $s(R)$. □

Substitutionen ersetzen einzelne Symbole durch Wortmengen, also in der Regel sogar durch unendlich viele Wörter, unter denen auch das leere Wort $\lambda$ vorkommen darf. Die Umkehrung, nämlich längere (Teil-)Wörter in einer Zeichenkette auf einzelne Symbole zu verkürzen, scheint zunächst nicht so leicht möglich. Mathematisch wird dies durch die Umkehrung eines Homomorphismus, also der Anwendung eines inversen Homomorphismus geleistet.

**3.64 Definition**

*Seien $X$ und $Y$ endliche Alphabete sowie $h : \Sigma^* \longrightarrow Y^*$ ein Homomorphismus. Die zu $h$ inverse Relation, genannt* ***inverser Homorphismus****, ist meist nur eine partielle Funktion $h^{-1} : Y^* \longrightarrow 2^{\Sigma^*}$. Hierbei definiert $h^{-1}$ für jedes $w \in Y^*$ die Menge aller möglichen Urbilder von $w$ durch:*

$$h^{-1}(w) := \{v \in \Sigma^* \mid \exists w \in Y^* : h(v) = w\}.$$

*Diese Menge kann daher auch leer sein. Wie auch bei Homomorphismen und Substitutionen wird der inverse Homomorphismus kanonisch auf formale Sprachen erweitert:*

$$h^{-1}(L) := \bigcup_{w \in L} h^{-1}(w)$$

**3.65 Beispiel (inverse Homomorphismen)**

Wir definieren zwei Homomorphismen und betrachten die Anwendung jeweils des inversen Homomorphismus auf (a) einzelne Wörter und (b) auf Sprachen. In (b) wird die kanonische Erweiterung in der üblichen Form verwendet.

(a) Sei $h : \{a, b, c\}^* \longrightarrow \{x, y\}^*$ definiert durch

$$\begin{array}{rcl} a & \mapsto & xyx, \\ b & \mapsto & xy, \\ c & \mapsto & yx. \end{array}$$

Dann ist $h^{-1}(xy) = \{b\}$, $h^{-1}(xx) = \emptyset$ und $h^{-1}(xyxyx) = \{ac, ba\}$.

(b) Sei $f : \{a, b, c\}^* \longrightarrow \{x, y\}^*$ definiert durch

$$\begin{array}{rcl} a & \mapsto & x, \\ b & \mapsto & y, \\ c & \mapsto & \lambda. \end{array}$$

Dann ist $h^{-1}(\{\lambda\}) = \{c\}^*$, $h^{-1}(\{y\}) = \{c\}^*\{b\}\{c\}^*$ und $h^{-1}(\{x, y\}^*) = \{a, b, c\}^*$.

**3.66 Theorem**

*Sei $L \in \mathcal{A}kz(Y)$ und $h : \Sigma^* \longrightarrow Y^*$ ein Homomorphismus, dann ist $h^{-1}(L) \in \mathcal{A}kz(\Sigma)$.*

**Beweis:** Sei $L = L(A)$ für einen vDFA $A = (Z, Y, \delta_1, z_0, Z_{\text{end}})$. Wir konstruieren einen DFA $B = (Z, \Sigma, \delta_2, z_0, Z_{\text{end}})$, der bei Eingabe eines Symbols $x$ aus $X$ den vDFA $A$ auf der Eingabe $h(x)$ simuliert. Dazu wird $\delta_2$ definiert durch: $\forall (z, x) \in Z \times X : \delta_2(z, x) := (z)^{h(x)}$, wobei in $B$ der mit $h(x)$ in $A$ erreichte Zustand $(z)^{h(x)}$ eingenommen wird. Wenn also in $A$ ein Wort $v \in (h(X))^*$ akzeptiert wird, so wird jedes $u \in \Sigma^*$ mit $h(u) \in (h(X))^*$ auch von $B$ akzeptiert. Die Umkehrung ist ebenso einfach. □

Wir hatten in Kapitel 2 (Def. 2.48) die Operation der Spiegelwortbildung $w^{\text{rev}}$ kennengelernt, um z.B. Palindrome definieren zu können und um den Zusammenhang zwischen rekursiven Definitionen und induktiven Beweisen zu illustrieren. Wir werden sehen, dass die Familie $\mathcal{R}eg$ auch gegenüber der Spiegelwortabbildung abgeschlossen ist:

**3.67 Theorem**

*Für jede reguläre Menge $L \in \mathcal{R}eg$ gilt $L^{\text{rev}} \in \mathcal{R}eg$,*

**Beweis:** Sei $L = L(A)$ für einen vDFA $A = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$. Wir konstruieren den buchstabierenden NFA $A_{\text{rev}} = (Z, \Sigma, K, Z_{\text{end}}, Z_{\text{start}})$ mit $K := \{(q, x, p) \mid \delta(p, x) = q\}$. Es ist offensichtlich, dass jeder Erfolgspfad in $A_{\text{rev}}$ einem rückwärts von einem End- zu einem Startzustand in $A$ gegangener Erfolgspfad entspricht. □

Die in den Theoremen 3.63, 3.66 und 3.67 behandelten Opearationen stellen also der Familie $\mathcal{R}eg$ weitere Abschlusseigenschaften zur Verfügung!

Eine oftmals sehr einfache Methode, von gewissen Mengen zu zeigen, dass sie nicht regulär sind, geht nach folgendem Schema vor:

Zunächst wird angenommen, dass die fragliche Menge regulär sei. Dann benutzt man Eigenschaften regulärer Mengen, um durch Transformationen mit Abschluss-Operatoren solche Sprachen zu erhalten, von denen man schon weiß, oder für die man vielleicht leichter zeigen kann, dass diese nicht regulär sind. Sind sie dieses nicht, so kann die Ausgangsmenge auch nicht regulär sein, denn wäre sie es, so doch auch alle Mengen, die aus dieser mit den Abschlußeigenschaften erzeugt wurden.

**3.68 Beispiel**

(a) Wir zeigen, dass die Menge $C := \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ nicht regulär ist, wie folgt: Setze $D := C \cap \{a\}^*\{b\}^*$. Mit $C \in \mathcal{REG}$ wäre auch $D \in \mathcal{REG}$, es ist aber $D = DUP = \{a^n b^n \mid n \in I\!N\}$ bekanntlich nicht regulär.

(b) Wir sahen, dass das $uvw$-Theorem nicht direkt benutzbar war, um zu beweisen, dass die Sprache $L := \{c^k a^l b^m \mid k,l,m \in I\!N : k = 0 \text{ oder } l = m\}$ nicht regulär ist. Wir definieren nun die Menge $E := L \cap \{c\}\{a\}^*\{b\}^*$, die wegen Satz 3.59 regulär wäre, wenn $L$ dies ist. Wenn wir nun auf $E$ den Homomorphismus $h : \{a,b,c\}^* \longrightarrow \{a,b\}^*$ anwenden, der das einzige „$c$" löscht und auf den anderen Symbolen die Identität bleibt, so gilt doch $h(E) = DUP \notin \mathcal{R}eg$ und folglich auch $L \notin \mathcal{R}eg$!

Neben den bisher kennengelernten werden wir nun noch eine weitere Abschlußeigenschaft der Sprachfamilie der regulären Mengen vorstellen.

**3.69 Definition**

*Für Mengen $K, L \subseteq \Sigma^*$ ist der* ***Rechtsquotient*** $K/L := \{u \in \Sigma^* \mid \exists v \in L : uv \in K\}$ *erklärt.*

**3.70 Theorem**

1. *Sind $L_1$ und $L_2$ reguläre Mengen, so ist auch $L_1/L_2$ eine reguläre Menge.*
2. *Ein DFA der diese Menge akzeptiert kann effektiv aus den üblichen Spezifikationen für die regulären Mengen $L_1$ und $L_2$ konstruiert werden.*

**Beweis:** Sei $A = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$ ein DFA der $L_1$ akzeptiert. In diesem ändern wir nun nur noch die Menge der Endzustände zu $Z'_{\text{end}} := \{z \in Z \mid \exists v \in L_2 : (z)^v \in Z_{\text{end}}\}$. Diese neue Menge lässt sich aber effektiv ausrechnen, denn für jeden Zustand $z \in Z$ ist die Leistung $L(z)$ regulär und mit ihr auch die Menge $L(z) \cap L_2$. Ob dieser Durchschnitt nun aber ein Element enthält oder nicht, lässt sich mit dem später genannten Theorem 3.74 entscheiden. □

Es sollte hier bemerkt werden, dass 1. in Theorem 3.70 auch noch richtig ist, wenn die Sprache $L_1$ regulär ist, jedoch $L_2$ eine beliebige Menge von Wörtern ist. Diese braucht dann also nicht mehr regulär zu sein! Der definierte endliche Automat ist immer noch funktionsfähig und leistet das Gewünschte. Es ist jedoch denkbar, dass diesen Automaten niemand wirklich hinschreiben kann, weil die Frage $L(z) \cap L_2 \neq \emptyset$ nicht für jede Sprache $L_2$ effektiv lösbar ist. Wir werden in späteren Veranstaltungen des Studiums, wie unter anderem in der Vorlesung F3 („Berechenbarkeit und Komplexität"), solche Mengen noch explizit kennenlernen.

## 3.10 Entscheidungsprobleme

Nach der Konstruktion von endlichen Automaten und dem Umgang mit diesen stellen sich viele Probleme als wichtig heraus. Zum Beispiel kann es nötig sein, zu erfahren, ob ein vorgegebenes Wort von diesem akzeptiert wird. Diese Frage muss zum Beispiel in einer ersten Phase der Syntaxanalyse imperativer (auch: zustandsorientierter) Programme, der lexiklaischen Analyse vom sogennannten „*scanner*" entschieden werden.

Für die Untersuchung in der theoretischen Informatik ist es praktisch, Probleme und Problem-Klassen stets in der gleichen Weise zu notieren. Wir werden diese hier im Folgenden vorstellen:

**Definition (⟨ Name oder Bezeichnung des Problems ⟩)**

**Eingabe:** ⟨ Auflistung der verwendbaren Daten ⟩
**Frage:** ⟨ Zu gewinnende neue Information ⟩

Die Frage nach einem Entscheidungsverfahren für eine reguläre Menge $R$, also nach einem Algorithmus, der zu jedem vorgegebenem Wort verkündet, ob dieses in der Menge $R$ enthalten ist oder nicht, wird als das (**spezielle**) **Wortproblem** der Menge $R$ bezeichnet. Die Spezifikation der Menge $R$ können wir uns beliebig denken, z.B. durch einen NFA oder einen rationalen Ausdruck, denn darauf kommt es nicht an. Es muss nur ein Algorithmus **existieren**, der das gewünschte leistet, konstruieren brauchen wir diesen aus der gegebenen Sprachspezifikation von $R$ nicht! In der verabredeten Schreibweise wird das (spezielle) Wortproblem also wie folgt spezifiziert:

**3.71 Definition (Wortproblem für eine reguläre Menge $L$)**
**Eingabe:** *Ein Wort* $w \in \Sigma^*$
**Frage:** *Gilt* $w \in L$*?*

Als Algorithmus wählen wir einfach den DFA der $R$ akzeptiert. Der zeitliche Aufwand zur Lösung dieses Problems ist damit nicht wesentlich größer als die Zeit, jedes einzelne Symbol des Wortes $w$ zu lesen und in der Übergangstafel des DFA nachzusehen. Der Aufwand ist also linear in der Länge des Wortes $w$, wenn davon ausgegangen wird, dass jeder einzelne Schritt eines Zustandsübergangs konstant viel Zeit benötigt.

Auch wenn diese Art der Problemstellung in solch einem einfachen Fall etwas künstlich erscheint, hat sie doch in späteren Fällen unbestreitbare Vorteile. Zum Beispiel dann, wenn wir ein Verfahren suchen, das nicht nur für diesen einen Automaten das Wortproblem entscheidet, sondern sogar für jeden beliebigen vorgelegten Automaten anwendbar ist. Diese Frage wird als das **allgemeine Wortproblem** bezeichnet und ist formulierbar als:

**3.72 Definition (Allgemeines Wortproblem für $\lambda$-freie NFA's)**
**Eingabe:** *Ein NFA* $A = (Z, \Sigma, K, z_0, Z_{\text{end}})$ *und ein Wort* $w \in \Sigma^*$
**Frage:** *Gilt* $w \in L(A)$*?*

Das allgemeine Wortproblem für NFA's ließe sich natürlich dadurch lösen, dass zuerst der NFA in einen äquivalenten DFA umgewandelt wird, und danach in diesem nachgesehen wird, ob der Pfad bei Eingabe von $w$ in einen Endzustand führt. Das ist wegen der möglichen Größe des DFA nicht zu empfehlen.

In vielen Fällen hilft es, wenn die Konstruktion des Potenzautomaten nur insoweit durchgeführt wird, wie es die möglichen Pfade in dem $\lambda$-freien NFA entlang des Wortes $w := w_1 w_2 w_3 \dots w_n$ erfordern.

Aber auch das sieht zunächst noch recht unangenehm aus, verlangt das Verfahren doch, dass von der Menge $M_i := \{q \in Z \mid \exists p \in Z_{\text{start}} : p \xrightarrow[v]{*} q, \text{ für } v = w_1 w_2 \dots w_i\}$ aller in $i$ Schritten aus der Menge $Z_{\text{start}}$ erreichbaren Zustände, im nächsten Schritt die folgende Menge $M_{i+1} := \{q \in Z \mid \exists p \in M_i : p \xrightarrow[w_{i+1}]{*} q\}$ konstruiert werden muss. Das benötigt in der Regel jedesmal $|M_i| \cdot |X|$ viele Schritte, also in der Summe im schlechtesten denkbaren Fall, wenn nämlich in jedem Schritt eine neue Teilmenge von $Z$ erzeugt wird, höchstens $|Z|^n$. Man betrachte die Automaten der Theoreme 3.29 und 3.50 und entsprechend gewählte Wörter.

Letztlich können diese Mengen jedoch nie mehr als $|Z|$ Elemente enthalten, so dass sich diese bei Wörtern mit $|w| > |Z|$ nicht mehr vergrößern und jeder Übergang höchsten $|Z|^2$ Schritte zum Nachsehen in der Übergangstabelle benötigt.

**3.73 Definition (Leerheitsproblem (emptiness problem) für endliche Automaten)**
**Eingabe:** *Ein DFA oder ein NFA $A$*
**Frage:** *Ist $L(A) = \emptyset$?*

**3.74 Theorem**
*Das Leerheitsproblem für endliche Automaten kann in linearer Zeit entschieden werden.*

**Beweis:** Gegeben sei ein NFA $A = (Z, \Sigma, K, Z_{\text{start}}, Z_{\text{end}})$. $L(A) = \emptyset$ genau dann, wenn es in dem Zustandsgraphen von $A$ von keinem der Anfangszustände einen Pfad zu einem Endzustand gibt. Das kann mit Tiefensuche (*depth-first*) und Zurücksetzen (*backtracking*) leicht getestet werden.

**3.75 Algorithmus**
Im Verlauf des Verfahrens werden Zustände ROT gefärbt. Zu Beginn ist noch kein Zustand gefärbt.

**0.** Man beginne bei einem der Startzustände $z_0$ und markiere diesen ROT.

**1.** Sei $z_1$ der aktuelle ROT gefärbte Zustand und $\{z_{1.1}, \dots, z_{1.k}\}$ die Menge der Folgezustände von $z_1$. (Beachte $k \leq |X|$).

**2.** Gibt es unter den Folgezuständen des aktuellen Zustands $z_1$ einen Endzustand, so gilt $L(A) \neq \emptyset$ und der Algorithmus stoppt. Andernfalls wähle man einen ungefärbten Folgezustand $z_{1.i}$ von $z_1$ zum aktuellen Zustand (das ist der Schritt der Tiefensuche) und färbt diesen ROT. Gehe zu **1.**
Gibt es keinen ungefärbten Folgezustand von $z_1$, so gehe man zu dem Vorgänger von $z_1$ (das ist das Zurücksetzen) von dem man diesen Zustand auf dem Hinweg erreicht hatte (dieser ist schon auf dem Hinweg ROT gefärbt worden) und gehe zu **1.**

Jede Kante des Zustandsgraphen wird auf diese Weise höchstens zweimal durchlaufen und dabei ein Baum aller vom Ausgangszustand erreichbaren Zustände aufgebaut. Wird das Verfahren für alle Startzustände durchgeführt, sind alle Zustände ROT gefärbt und die Antwort erhalten. Der Gesamtaufwand wird durch die Anzahl der existierenden Kanten linear begrenzt und die Größe des endlichen Automaten ist ebenfalls linear in der Anzahl der Kanten. □

Das Äquivalenzproblem für DFA ist gegeben durch:

**3.76 Definition (Äquivalenzproblem für DFA's)**
**Eingabe:** *Zwei DFA* $A = (Z_A, X_A, \delta_A, \{z_{A,0}\}, Z_{A,end})$ *und* $B = (Z_B, X_B, \delta_B, \{z_{B,0}\}, Z_{B,end})$
**Frage:** *Gilt* $L(A) = L(B)$*?*

**3.77 Theorem**
*Das Äquivalenzproblem für DFA ist in Zeit* $|X| \cdot |Z| \cdot |Z'|$ *entscheidbar.*

**Beweis:** Für $L_1 := L(A)$ und $L_2 := L(B)$ gilt doch $L_1 = L_2$ genau dann, wenn sowohl $L_1 \cap \overline{L_2} = \emptyset$ als auch $L_2 \cap \overline{L_1} = \emptyset$ ist. Ein vervollständigter DFA kann sofort in einen DFA umgewandelt werden, der das Komplement des ersten akzeptiert. Dazu müssen nur alle ehemaligen Endzustände zu gewöhnlichen Zuständen modifiziert und umgekehrt alle Zustände, die keine Endzustände waren, nun in Endzustände gewandelt werden. Dies geschieht mit linearem Aufwand. Den Durchschnittsautomaten $C_{A\cap B} := (Z_A \times Z_B, X_3 = X_A \cap X_B, \delta_3, \{(z_{A,0}, z_{B,0})\}, Z_{3,\text{end}})$ hatten wir schon im Beweis zu Theorem 3.59 kennengelernt. Seine Konstruktion verlangt die Bestimmung aller Einträge der Übergangsfunktion $\delta_3$ mit $|X_3| \cdot |Z_A| \cdot |Z_B|$ vielen Argumenten in entsprechend vielen Schritten. □

**3.78 Definition (Universalitätsproblem für DFA's)**
**Eingabe:** *Ein DFA* $A = (Z, \Sigma, \delta, z_0, Z_{end})$
**Frage:** *Ist* $L(A) = \Sigma^*$*?*

**3.79 Theorem**
*Das Universalitätsproblem für deterministische endliche Automaten ist in linearer Zeit entscheidbar.*

**Beweis:** Gegeben sei ein DFA $A = (Z, \Sigma, \delta, z_0, Z_{\text{end}})$. Es gilt $L(A) = \Sigma^*$ genau dann, wenn es in dem Zustandsgraphen von $A$ von dem Anfangszustand keinen Pfad zu einem Zustand gibt, der nicht Endzustand ist und wenn außerdem für jeden Zustand und jedes Eingabesymbol ein Nachfolgezustand definiert ist. Das kann leicht festgestellt werden, denn dann muss jeder Zustand ein Endzustand sein und jeden Zustand müssen $|\Sigma|$ Kanten verlassen! □

# A Historischer Ursprung

Die Bezeichnung „Algorithmus“ für ein nach festen Regeln ablaufendes Verfahren, geht auf den persischen Mathematiker Ibn Musa Al Chwarizmi (al-Khowarizmi) zurück, der etwa um 820 n. Chr. das Buch *Kitab al jabr w'almuqabala* („Regeln der Wiedereinsetzung und Reduktion“) über die Behandlung algebraischer Gleichungen schrieb. Um 1300 suchte der katalanische Mystiker, Dichter und Missionar Ramón Llull (sprich: „Jui“, bekannt als Raimundus Lullus, 1232 – 1316) nach einer allgemeinen Methode zum Auffinden aller „Wahrheiten“ auf kombinatorischer Grundlage. Mathematisch von geringer Bedeutung, beeinflußte sein Werk, die *ars magna*, viele spätere Mathematiker. Unter anderen den italienischen Philosophen, Arzt und Mathematiker Geronimo Cardano (1501 – 1576), René Descartes (Renatus Cartesius, 1596 – 1650) und Gottfried Wilhelm Leibnitz (1646 – 1716).

Leibnitz kann man, wegen seiner Überlegungen, Ideen und Beiträge, als den eigentlichen Vater der (Theoretischen) Informatik bezeichnen, obwohl seine Schriften keinen unmittelbaren Einfluss auf die Entwicklung der mathematischen Logik hatte. In seiner Schrift: *Explication de l'Arithme'tique Binaire* beschreibt er das Rechnen im dualen Zahlensystem und hatte wohl als erster die Idee zu einer Rechenmaschine auf der Basis des Dualsystems. Beeinflußt von Lullus, drückt Leibnitz in dem Werk *De scientia universalis seu calculo philosophico* seine Ideen zur symbolischen Darstellung von Begriffsinhalten aus. Er vermutete, dass eines Tages jedes wissenschaftliche Streitgespräch von Philosophen durch „Ausrechnen“ entschieden werden könne.

Weniger philosophisch/mathematisch orientierte Manipulationen mit Symbolen traten schon früh bei dem Gebrauch von Geheimschriften auf. Die einfachen Transpositionen von Julius Ceasar sind, vom Standpunkt der heutigen Kryptologie, kaum ernstzunehmen. Das erste erhaltene Werk zur Kryptologie stammt von Leon Battista Alberti (ital. Baumeister und Kunsthistoriker, 1404 – 1472).

Von einer anderen Seite kommend suchten die Logiker nach möglichst universell anwendbaren Algorithmen. Die auf Aristoteles (384-322 v.Chr.) zurückgehende Syllogistik umfaßt nur einen kleinen Teil der für Mathematiker relevanten Schlüsse. Leibnitz beschrieb (wieder als erster) 1672 – 1676 die wesentlichen Ansätze für einen logischen Kalkül. Mit einer präzisen Universalsprache, der *characteristica universalis*, wollte er Aussagen beschreiben und mit dem Kalkül, dem *calculus ratiocinator*, weiter behandeln. Noch heute folgt man in wesentlichen Teilen dieser Methode zur Definition von Kalkülen, wie in diesem Skript im Detail ausgeführt und an einigen Beispielen erläutert wurde.

Der Schweizer Mathematiker Leonard Euler (1707 – 1783) wie auch der Engländer John Venn gaben Diagramme zur Erklärung und Durchführung logischer Operationen. Zu den Wegbereitern der modernen Logik zählt vor allem der Engländer George Boole (1815 – 1864) und sein Kollege Augustus de Morgan (1806 – 1871). Boole's algebraische Sichtweise der (Aussagen-)Logik wurde später, besonders durch G.

Peano (1858 – 1932), mehr und mehr durch eine, der natürlichen Sprache angepaßte, Symbolik ersetzt. Mit seiner *Begriffsschrift* bemühte sich Gottlob Frege (1848 – 1925) um eine exakte Fassung der logischen Regeln. Alfred North Whitehead (1861 – 1949) und Bertrand Russell (1872 – 1970) formulierten mit dem monumentalen Werk: *Principia Mathematica* eine logische Grundlage der Mathematik. Auf der Suche nach Beweisen für den Prädikatenkalkül erster Stufe führten die Arbeiten von Skolem, etwa 1920, 1930 von Herbrand, und 1933 von Gerhard Gentzen (1909 – 1945) sowie 1956 von J.A. Robinson (1930 – ) zu Formalen Systemen mit den Begriffen von Folgerbarkeit und Ableitbarkeit. Einen krönenden Abschluß fand die vorangegangene Entwickung 1930 mit dem Gödel'schen Vollständigkeitssatz für die Prädikatenlogik erster Stufe und 1931 mit dem Beweis dafür, dass jede widerspruchsfreie Axiomatisierung der Zahlentheorie nicht entscheidbare Aussagen enthält. Das heißt, dass es eine nicht leere Menge von wahren Aussagen gibt, für die es jedoch kein Verfahren gibt, einen Beweis dieser Tatsache für alle Aussagen dieser Menge zu finden.

Die Suche nach einer Klärung des Begriffs „Algorithmus" ist aufs engste verbunden mit den Versuchen einer formalen Definition der als intuitive berechenbar angesehenen Funktionen. Auf David Hilbert (1862 – 1943) und P. Bernays geht der Begriff der primitiven Rekursion zurück, während Stephen C. Kleene die Bezeichnung der primitiv rekursiven Funktionen prägte, die 1931 von Kurt Gödel (1906 – 1978) noch als rekursive Funktionen bezeichnet wurden. W. Ackermann beschrieb 1928 eine offensichtlich intuitiv berechenbare Funktion, die nicht primitiv rekursiv ist und zeigte so, dass dieser Begriff nicht alle intuitiv berechenbaren Funktionen umfaßt. 1936 schlug Alonzo Church (1903 – 1994) vor, den Begriff der $\lambda$-definierbaren Funktionen mit dem der intuitiv berechenbaren Funktionen zu identifizieren. Dies wird heute als Church'sche These bezeichnet. Da diese Funktionen nicht überall definiert sein müssen, werden sie heute meist als partiell rekursiv bezeichnet.

Präzisierungen des Algorithmusbegriffs wurden mit unterschiedlichen formalen Methoden etwa gleichzeitig entwickelt: J. Herbrand 1932, Kurt Gödel 1934 und Stephen C. Kleene 1936 benutzten den Gleichungskalkül, Alonzo Church 1936 den Lambda-Kalkül, Alan M. Turing (1912 – 1954) formulierte 1937 die jetzt nach ihm benannten Maschinen und Emil L. Post (1887 – 1954) definierte 1936 formale Wortersetzungssysteme. Axel Thue (1863 – 1922) definierte 1906 die sogenannten Thue-Systeme und 1914 die heutigen semiThue-Systeme. Alle diese Präzisierungen haben sich als äquivalent erwiesen und beschreiben das, was die Erfinder als intuitiv berechenbar ansahen. Die Thesen, alles intuitiv Berechenbare sei mit dem jeweiligen Kalkül erfaßbar, werden jetzt als Turing'sche beziehungsweise Church'sche These bezeichnet. Tatsächlich stellten sich alle weiteren formulierten Definitionen des Algorithmus-Begriffs immer wieder als äquivalent zu den vorgenannten heraus. Damit fand diese Entwicklung einen bis heute noch gültigen Abschluß.

# Literaturverzeichnis


[Aho&Ullman] A.V. Aho & J.D. Ullman:
"The Theory of Parsing, Translation and Compiling",
Prentice Hall, vol I (1972), vol II (1973)

[Aho,Hopcroft&Ullman] A.V. Aho, J.E. Hopcroft & J.D. Ullman:
"The Design and Analysis of Computer Algorithms",
Addison Wesley, Reading Mass. (1974)

[Aho,Sethi&Ullman] A.V. Aho, R. Sethi & J.D. Ullman:
"Compilers – Principles, Techniques, and Tools",
Addison Wesley, Reading Mass. (1986)

[Aigner] M. Aigner:
"Diskrete Mathematik",
Friedr. Vieweg & Sohn, Verlagsgesellschaft mbH., Braunschweig, Wiesbaden (1993)

[Baase] S. Baase:
"Computer Algorithms
– Introduction to Design and Analysis",
Addison Wesley, Reading Mass. (1988)

[Biggs] N.L. Biggs:
"Discrete Mathematics"
Oxford University Press Inc. New York (1998)

[Brauer] W. Brauer:
"Automatentheorie",
Teubner, Stuttgart (1984)

[Bronstein et al.] I.N. Bronstein & K.A. Semedjajew & G. Musiol & H. Mühlig:
"Taschenbuch der Mathematik"
Verlag Harri Deutsch, Frankfurth a.M. (1995)

[Duden Informatik] "Duden Informatik"
Dudenverlag, Mannheim (1993)

[Floyd&Beigel] R. Floyd & R. Beigel:
"Die Sprache der Maschinen",
Internat. Thomson Publ., Bonn, Albany (1996)

[Graham et al.] R.L. Graham, D.E. Knuth, & O. Patashnik:
"Concrete Mathematics",
Addison-Wesley, Reading, Ma. (1989)

[Gries&Schneider] D. Gries & F.B. Schneider:
"A Logical Approach to Discrete Math".
Springer-Verlag, Heidelberg (1993)

[Gruska] J. Gruska:
"foundations of computing".
Int. Thomson Computer Press, London, Boston, Bonn (1997)

[Foster] J.M. Foster:
"Automatische Syntax-Analyse",
Carl Hanser Verlag, München (1971)

[Harrison] M.A. Harrison:
"Introduction to Formal Language Theory",
Addison-Wesley, Reading Mass. (1978)

[Hopcroft,Motwani&Ullman] J.E. Hopcroft, R. Motwani & J.D. Ullman:
"Introduction to Automata Theory, Languages, and Computation",
Addison-Wesley, 2nd edition (2001)

[Hopcroft&Ullman] J.E. Hopcroft & J.D. Ullman:
"Introduction to Automata Theory, Languages, and Computation",
Addison-Wesley, Reading Mass. (1979)

[Kinber&Smith] E. Kinber & C. Smith:
"Theory of Computing – A Gentle Introduction",
Prentice Hall, Inc. (2001)

[Loeckx,Mehlhorn&Wilhelm] J. Loeckx, K. Mehlhorn & R. Wilhelm:
"Grundlagen der Programmiersprachen",
Teubner, Stuttgart (1986)

[Maurer] H. Maurer:
"Theoretische Grundlagen der Programmiersprachen",
B.I. Wissenschaftsverlag, Stuttgart (1969)

[Möller] H. Möller:
"Algorithmische Lineare Algebra – Eine Einführung für Mathematiker und Informatiker",
Friedr. Vieweg & Sohn, Verlagsgesellschaft mbH., Braunschweig, Wiesbaden (1997)

[Sander,Stucky&Herschel] P. Sander, W. Stucky & R. Herschel:
"Automaten, Sprachen, Berechenbarkeit",
Teubner, Stuttgart (1992)

[Schöning] U. Schöning:
"Theoretische Informatik kurz gefasst",
B.I. Wissenschaftsverlag, Mannheim (1992)

[Turing] A.M. Turing:
"On computable numbers with an application to the Entscheidungsproblem",
Proc. London Math. Soc. 42 (1936) 230-265,
43 (1937) 544-546

[Wegener] I. Wegener:
"Theoretische Informatik",
B.G. Teubener, Stuttgart (1993)

[Zobel] R. Zobel:
"Diskrete Strukturen – eine angewandte Algebra für Informatiker",
Reihe Informatik Band 49,
B.I. Wissenschaftsverlag, Mannheim (1987)