Skript zur Vorlesung

Logik und diskrete Strukturen

Prof. Dr. Heiko Röglin
Institut für Informatik



2. Februar 2024

Vorwort

Dieses Skript ist als Begleitmaterial für die Vorlesung „Logik und diskrete Strukturen“ an der Universität Bonn konzipiert.

Ich danke Tobias Brunsch für das Korrekturlesen des Skriptes. Ebenso danke ich den zahlreichen Studenten, die durch ihre Anmerkungen dazu beigetragen haben, das Skript zu verbessern. Für weitere Hinweise auf Fehler und Verbesserungsvorschläge bin ich jederzeit dankbar. Bitte senden Sie diese an meine E-Mail-Adresse `roeglin@cs.uni-bonn.de` oder sprechen Sie mich in der Vorlesung an.

Heiko Röglin

Inhaltsverzeichnis

1 Einleitung **5**

2 Mathematische Grundlagen **9**

2.1 Mengen 9

2.2 Beweise 14

2.2.1 Aussagen 14

2.2.2 Implikationen und Äquivalenzen 15

2.2.3 Direkte und indirekte Beweise 17

2.2.4 Vollständige Induktion 19

2.3 Quantoren 22

2.4 Relationen und Abbildungen 24

2.4.1 Relationen 24

2.4.2 Abbildungen 26

2.4.3 Äquivalenzrelationen 29

3 Endliche Automaten und formale Sprachen **33**

3.1 Sprachen und Grammatiken 34

3.2 Endliche Automaten 37

3.2.1 Pumping-Lemma für endliche Automaten 39

3.2.2 Das Pumping-Lemma als Spiel 41

3.2.3 Nichtdeterministische endliche Automaten 43

3.2.4 Minimierung endlicher Automaten 46

3.3 Reguläre Sprachen, endliche Automaten und reguläre Ausdrücke 50

4 Ausgewählte Themen der Mathematik **58**

4.1 Abzählbare und überabzählbare Mengen 58

4.2 Abzählende Kombinatorik 63

4.3 Algebraische Strukturen 70

4.3.1 Halbgruppen, Monoide und Gruppen 73

4.3.2 Ringe und Körper 75

4.3.3 Euklidischer Algorithmus 78

4.3.4 Chinesischer Restsatz 82

4.3.5 RSA-Kryptosystem 85

5 Einführung in die mathematische Logik **92**

5.1 Aussagenlogik 93

5.1.1 Syntax 93

5.1.2 Semantik 95

5.1.3 Normalformen 99

5.1.4 Resolutionskalkül 104

5.2 Prädikatenlogik 110

5.2.1 Signaturen und Strukturen 110

5.2.2 Syntax 112

5.2.3 Semantik 114

5.2.4 Ausblick 118

Kapitel 1

Einleitung

Herzlich willkommen zu der Vorlesung *Logik und diskrete Strukturen*! Es handelt sich hierbei um die erste von drei Pflichtvorlesungen im Bereich der *theoretischen Informatik*, die im Bachelorstudiengang an der Universität Bonn vorgesehen sind. Bevor es richtig los geht, geben wir zunächst einen Überblick, worum es in der theoretischen Informatik geht und was Sie in dieser Vorlesung und den Vorlesungen *Algorithmen und Berechnungskomplexität I und II* im dritten und vierten Semester erwartet.

Die theoretische Informatik ist ein so vielfältiges Forschungsfeld, dass wir statt einer Definition lieber einige ihrer Teildisziplinen vorstellen. Diese geben einen repräsentativen Überblick und motivieren Sie hoffentlich dazu, sich eingehender mit diesem spannenden Thema auseinanderzusetzen.

- Die *Algorithmik* beschäftigt sich mit dem Entwurf und der Analyse von *Algorithmen*. Ein Algorithmus ist eine Handlungsvorschrift zur Lösung eines Problems, die so präzise formuliert ist, dass sie von einem Computer ausgeführt werden kann. Algorithmen begegnen uns ständig im täglichen Leben, ohne dass wir sie bewusst wahrnehmen. Navigationsgeräte bestimmen den besten Weg vom Start zum Ziel, Suchmaschinen durchsuchen innerhalb kürzester Zeit riesengroße Datenmengen und beim Onlinebanking werden die übertragenen Daten ver- und entschlüsselt. Ohne clevere Algorithmen wären all diese Dinge unmöglich.

  Die Algorithmik beschäftigt sich damit, Algorithmen für verschiedene Probleme zu entwerfen. Die obigen Beispiele zeigen bereits, dass es oft eine ganz wesentliche Herausforderung ist, nicht nur korrekte, sondern auch effiziente Algorithmen zu finden, die so schnell wie möglich das richtige Ergebnis liefern. Um dies zu ermöglichen und verschiedene Algorithmen miteinander vergleichen zu können, werden Algorithmen theoretisch und experimentell analysiert. Diese Analysen liefern oft neue Einsichten, mit deren Hilfe verbesserte Algorithmen entworfen werden.

  In der Vorlesung *Algorithmen und Berechnungskomplexität I* werden wir Methoden zum Entwurf und zur Analyse von Algorithmen kennenlernen. Diese werden wir nutzen, um effiziente Algorithmen für grundlegende Probleme wie Sortieren, Suchen, und die Berechnung kürzester Wege zu entwerfen. Für die effiziente

Lösung dieser Probleme benötigt man nicht nur geeignete Algorithmen, sondern auch die richtigen *Datenstrukturen*, in denen die Daten gespeichert werden. Auch mit diesen werden wir uns beschäftigen und elementare Datenstrukturen wie Arrays, verkettete Listen, Suchbäume und Hashtabellen kennenlernen und analysieren.

- Die *Berechenbarkeitstheorie* und *Komplexitätstheorie* bilden das Gegenstück zur Algorithmik. Die Berechenbarkeitstheorie beschäftigt sich mit der Frage, welche Probleme überhaupt von Computern gelöst werden können, und die Komplexitätstheorie beschäftigt sich damit, welche Ressourcen (Rechenzeit, Speicherplatz etc.) notwendig sind, um bestimmte Probleme zu lösen.

  Zwar werden die verfügbaren Rechner immer leistungsfähiger, aber es gibt fundamentale Barrieren, die auch mit verbesserter Hardware nicht durchbrochen werden können. Haben Sie beispielsweise ein Programm geschrieben, das ein gewisses Problem lösen soll wie die Berechnung eines kürzesten Weges, dann ist es interessant, ob Ihr Programm immer terminiert oder ob es in eine Endlosschleife geraten kann. Ist Letzteres der Fall, so ist das Programm fehlerhaft und sollte überarbeitet werden. Es wäre also wünschenswert, wenn der Compiler in solchen Fällen direkt eine Warnung ausgeben würde. Tatsächlich kann man aber beweisen, dass es keinen Algorithmus gibt, der für beliebige Programme korrekt feststellt, ob eine Endlosschleife auftreten kann oder nicht. Das heißt, egal wie leistungsfähig unsere Rechner auch sein mögen, es wird nie einen Compiler geben, der dieses sogenannte *Halteproblem* löst.

  Die meisten Probleme, denen wir in der Informatik begegnen, können algorithmisch gelöst werden. Allerdings nutzen uns Algorithmen wenig, wenn sie nicht effizient sind. Wer möchte schon einen Tag auf die Ausgabe seines Navigationsgerätes warten? Leider gibt es aber eine ganze Reihe von Problemen, für die es vermutlich keine effizienten Algorithmen gibt. Ein Problem, das in vielen logistischen Anwendungen eine Rolle spielt, ist das Problem des Handlungsreisenden, bei dem eine Landkarte mit mehreren Städten gegeben ist und die kürzeste Rundreise durch diese Städte gesucht wird. Dieses Problem gehört wie viele andere einfach zu formulierende Probleme zu der Klasse der *NP-schweren* Probleme. Man vermutet, dass es für diese Probleme keine effizienten Algorithmen gibt. Diese sogenannte *P≠NP-Vermutung* ist bis heute unbewiesen und eines der größten ungelösten Probleme der Mathematik und theoretischen Informatik.

  In der Vorlesung *Algorithmen und Berechnungskomplexität II* werden wir eine Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie geben und uns ausführlich mit nicht berechenbaren Problemen wie dem Halteproblem und NP-schweren Problemen wie dem Problem des Handlungsreisenden beschäftigen. Zwar werden in diesem Themenbereich hauptsächlich negative Ergebnisse gezeigt, diese haben aber wichtige Auswirkungen auf die Praxis. Hat man beispielsweise bewiesen, dass ein Problem nicht effizient gelöst werden kann, so ist klar, dass die Suche nach einem effizienten Algorithmus eingestellt werden kann und dass man stattdessen über Alternativen (Abwandlung des Problems etc.) nachdenken sollte. Außerdem beruhen Kryptosysteme auf der Annahme, dass gewisse Probleme nicht effizient gelöst werden können. Wäre es beispiels-

weise möglich, große Zahlen effizient zu faktorisieren, so wäre RSA kein sicheres Kryptosystem. In diesem Sinne können also auch negative Ergebnisse gute Nachrichten sein.

- *Automatentheorie und formale Sprachen* bilden die Grundlagen für den Entwurf von Programmiersprachen und den Compilerbau. Eine formale Sprache ist eine Beschreibung, welche Zeichenfolgen gültige Programme darstellen. Normalerweise erfolgt diese Beschreibung durch *Grammatiken*. Dabei handelt es sich um Regelsysteme, die beschreiben, wie syntaktisch korrekte Programme erzeugt werden. Die Theorie formaler Sprachen beschäftigt sich damit, wie diese Regelsysteme aussehen müssen, damit ein Compiler möglichst effizient die *lexikalische Analyse* und die *syntaktische Analyse* durchführen kann. Die lexikalische Analyse ist der erste Schritt, den ein Compiler durchführt; dabei wird der Quelltext in logisch zusammenhängende Tokens wie zum Beispiel Schlüsselwörter, Zahlen und Operatoren zerlegt. Der zweite Schritt ist die syntaktische Analyse, in der überprüft wird, ob der Quelltext ein syntaktisch korrektes Programm ist, und in der der Quelltext in einen sogenannten Syntaxbaum umgewandelt wird.

  Endliche Automaten sind einfache abstrakte Modelle von Rechnern, die weniger mächtig sind als herkömmliche Computer. Endliche Automaten werden eingesetzt, um für bestimmte formale Sprachen zu entscheiden, welche Wörter Teil der Sprache sind und welche nicht. Sie werden in Compilern bei der lexikalischen Analyse eingesetzt und sie spielen bei der Durchsuchung umfangreicher Texte nach bestimmten Wörtern und Mustern eine Rolle.

  In dieser Vorlesung werden wir uns mit formalen Sprachen und Automaten beschäftigen. Wir werden diskutieren, welche formalen Sprachen als Programmiersprachen geeignet sind und welche von endlichen Automaten entschieden werden können. Im dritten Semester werden wir diese Thematik weiter vertiefen.

- Die *Logik* beschäftigt sich damit, wie man formal Schlüsse zieht und Beweise führt. In der einfachsten Form, der *Aussagenlogik*, betrachtet man Aussagen, die wahr oder falsch sein können. Man geht davon aus, dass man von gewissen Elementaraussagen weiß, ob sie wahr oder falsch sind, und studiert dann die Wahrheitswerte von Aussagen, die durch Verknüpfungen (Verneinung, Konjunktion, Implikation, etc.) aus diesen Grundaussagen entstehen. Die Aussagenlogik ist nicht besonders ausdrucksstark (d. h. viele interessante Sachverhalte können mithilfe der vorhandenen Verknüpfungen nicht ausgedrückt werden), sie bildet aber die Grundlage für kompliziertere Logiken und eignet sich gut als Einstieg und zur Illustration wichtiger Konzepte wie der Trennung von Syntax und Semantik.

  Eine Erweiterung der Aussagenlogik ist die *Prädikatenlogik erster Stufe*. Ein wesentlicher Aspekt dieser Erweiterung ist, dass zusätzlich zu den vorhandenen Verknüpfungen der Aussagenlogik Quantoren erlaubt sind. Damit können Aussagen der Form „für alle x gilt ...“ und „es gibt ein x, für das gilt ...“ gebildet werden.

  Das Studium der Logik ist zwar eher der Mathematik als der Informatik zuzuordnen, wir beschäftigen uns aber aus zwei Gründen auch in der Informatik damit.

> Zum einen müssen unsere Analysen und Argumentationen denselben rigorosen Ansprüchen genügen wie Beweise in der Mathematik, weshalb es unumgänglich ist, dass wir uns damit beschäftigen, was überhaupt ein formal korrekter Beweis ist. Zum anderen gibt es eine Beziehung zu Datenbanken. Eine SQL-Anfrage ist im Wesentlichen eine Formel der Prädikatenlogik erster Stufe. Um also ein tieferes Verständnis von relationalen Datenbanken zu entwickeln und zu verstehen, wie mächtig SQL-Anfragen sind, ist das Studium der Prädikatenlogik äußerst hilfreich.

Die wesentlichen Themen dieser Vorlesung sind Logik, Automatentheorie und formale Sprachen. Zunächst werden wir aber einige mathematische Grundlagen besprechen, die für ein Studium der Informatik unerlässlich sind. Manche davon sind Ihnen wahrscheinlich bereits in der Schule begegnet. Dennoch werden wir uns hier die Zeit nehmen, sie zu wiederholen und zu vertiefen, da die sichere Beherrschung dieser Grundlagen eine wichtige Voraussetzung für jede Lehrveranstaltung der Informatik ist.

Dieses Skript umfasst die Inhalte der Vorlesung. Lesern, die an weitergehenden Themen interessiert sind oder die Themen dieser Vorlesung noch einmal aus einem anderen Blickwinkel studieren möchten, seien die Skripte der vergangenen Jahre [6, 2, 5] und die Bücher von Uwe Schöning [8, 9] sowie von Hopcroft, Motwani und Ullman [4] empfohlen. Viele Themen und Beispiele in diesem Skript sind diesen Quellen entnommen.

Kapitel 2

Mathematische Grundlagen

In diesem Kapitel besprechen wir einige mathematische Grundlagen, die für ein erfolgreiches Informatikstudium benötigt werden.

2.1 Mengen

Wir werden in dieser Vorlesung unter einer *Menge* immer eine Ansammlung von Objekten verstehen. So bilden beispielsweise die natürlichen Zahlen eine Menge ebenso wie alle Studenten der Informatik an der Universität Bonn. Dieser naive Mengenbegriff kann zu Paradoxa führen, die die Entwicklung der axiomatischen Mengenlehre in der Mathematik motiviert haben. Damit werden wir uns hier aber nicht beschäftigen, da für unsere Zwecke der naive Mengenbegriff zunächst ausreichend ist.

Ist M eine Menge und ist x ein Objekt in dieser Menge, so sagen wir, dass x ein *Element* der Menge M ist. Mengen, die nur aus endlich vielen Elementen bestehen, können wir durch die explizite Aufzählung ihrer Elemente beschreiben. Dabei werden die verschiedenen Elemente durch Kommas getrennt und die Elemente werden durch geschweifte Klammern eingeschlossen. Enthält die Menge M beispielsweise alle natürlichen Zahlen, die kleiner als 8 sind, so schreiben wir $M = \{1, 2, 3, 4, 5, 6, 7\}$. Oft deuten wir die Fortsetzung eines klar zu erkennenden Musters durch Auslassungspunkte an. Wir schreiben also beispielsweise abkürzend $M = \{1, 2, 3, \ldots, 7\}$ oder sogar $M = \{1, \ldots, 7\}$.

Wichtige Mengen von Zahlen

- Menge der *natürlichen Zahlen*: $\mathbb{N} = \{1, 2, 3, \ldots\}$
- Menge der natürlichen Zahlen mit Null: $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$
- Menge der *ganzen Zahlen*: $\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3, \ldots\}$
- Menge der *rationalen Zahlen*: $\mathbb{Q}$
- Menge der *reellen Zahlen*: $\mathbb{R}$

Ist x ein Element der Menge M, so schreiben wir $x \in M$. Ist x ein Objekt, das kein Element der Menge M ist, so schreiben wir $x \notin M$. Es gilt also zum Beispiel $4 \in \mathbb{N}$, $-4 \notin \mathbb{N}$, $\pi \in \mathbb{R}$ und $\pi \notin \mathbb{Z}$.

Anstatt alle Elemente einer Menge explizit anzugeben, können wir Mengen auch über die Eigenschaften ihrer Elemente definieren. Ist N eine Menge und A eine Eigenschaft, die manche Elemente in N besitzen, so ist

$$M = \{x \in N \mid x \text{ besitzt Eigenschaft } A\}$$

die Menge aller Elemente von N, die die Eigenschaft A besitzen. Geht die Menge N aus dem Kontext hervor, so verzichten wir oft auf ihre explizite Nennung.

Beispiele

- $P = \{n \in \mathbb{N} \mid n \text{ ist eine Primzahl}\}$
- $P' = \{n \in \mathbb{N} \mid n \text{ ist eine Primzahl und kleiner als } 15\}$
- $F =$ Menge aller Unicode-Zeichenketten
  $J = \{x \in F \mid x \text{ ist syntaktisch korrekte Java-Klasse}\}$
- $\mathbb{Q} = \left\{x \in \mathbb{R} \;\middle|\; \text{es gibt } a \in \mathbb{Z} \text{ und } b \in \mathbb{Z} \text{ mit } b \neq 0 \text{ und } x = \frac{a}{b}\right\}$

Wir sagen, dass eine Menge A eine *Teilmenge* von einer Menge B ist, wenn jedes Element von A auch in B enthalten ist. Dies schreiben wir kurz als $A \subseteq B$. Wir nennen $\subseteq$ auch eine *Inklusion*. Ist A eine Teilmenge von B, so schreiben wir manchmal statt $A \subseteq B$ auch $B \supseteq A$ und wir nennen B eine *Obermenge* von A. Es gilt beispielsweise $\mathbb{N} \subseteq \mathbb{N}_0$ und $\mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$. Ebenso gilt $M \subseteq M$ für jede Menge M und $J \subseteq F$ für die oben definierten Mengen J und F.

Häufig vorkommende Teilmengen von $\mathbb{R}$

- Für $a, b \in \mathbb{R}$ sei $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$.
  Wir nennen $[a, b]$ ein *abgeschlossenes Intervall*.
- Für $a, b \in \mathbb{R}$ sei $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$.
  Wir nennen (a, b) ein *offenes Intervall*.
- Es sei $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$ und $\mathbb{R}_{\leq 0} = \{x \in \mathbb{R} \mid x \leq 0\}$.

Wir sagen, dass zwei Mengen A und B *gleich* sind, wenn sie dieselben Elemente enthalten, und schreiben dann $A = B$. Außerdem benutzen wir $\emptyset$ als Symbol für die *leere Menge*, d. h. für die Menge, die kein Element enthält. Für jedes Objekt x gilt $x \notin \emptyset$ und für jede Menge M gilt $\emptyset \subseteq M$.

Beispiele

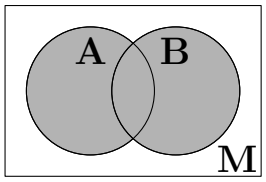
- $\{1, 2, 3\} = \{2, 3, 1\}$
- $\{n \in \mathbb{N} \mid n \text{ ist eine Primzahl und kleiner als } 15\} = \{2, 3, 5, 7, 11, 13\}$
- $\{n \in \mathbb{N} \mid n \text{ ist eine Zweierpotenz und durch 3 teilbar}\} = \emptyset$

Zwei Mengen A und B sind genau dann gleich, wenn $A \subseteq B$ und $B \subseteq A$ gilt. Um zu zeigen, dass zwei Mengen gleich sind, ist es oft die einfachste Möglichkeit, diese beiden Inklusionen getrennt voneinander nachzuweisen.

Es sei M eine Menge und es seien $A \subseteq M$ und $B \subseteq M$ Teilmengen von M. Wir definieren nun drei Möglichkeiten, wie man die Mengen A und B verknüpfen kann. In den sogenannten *Venn-Diagrammen* auf der rechten Seite sind die Mengen, die sich aus A und B ergeben, in grau dargestellt.

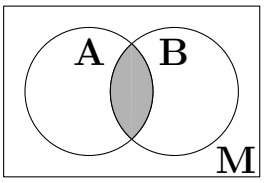
- Wir bezeichnen mit $A \cup B$ die *Vereinigung* von A und B, d. h.

$$A \cup B = \{x \in M \mid x \in A \text{ oder}^1 \ x \in B\}.$$



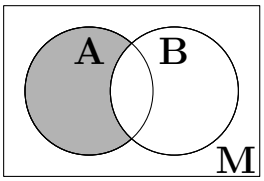
- Wir bezeichnen mit $A \cap B$ den *Durchschnitt* von A und B, d. h.

$$A \cap B = \{x \in M \mid x \in A \text{ und } x \in B\}.$$



- Wir bezeichnen mit $A \setminus B$ die *Differenz* von A und B, d. h.

$$A \setminus B = \{x \in M \mid x \in A \text{ und } x \notin B\}.$$



Beispiele

$\{1,3,5,7\} \cap \{2,5,8,9\} = \{5\}$	$\mathbb{N} \cap \mathbb{N}_0 = \mathbb{N}$
$\{1,2,3\} \cup \{2,3,4\} = \{1,2,3,4\}$	$\mathbb{N} \cup \mathbb{N}_0 = \mathbb{N}_0$
$\{1,2,3\} \setminus \{2,3,4\} = \{1\}$	$(\mathbb{R} \cap \mathbb{Z}) \setminus \{3\} = \mathbb{Z} \setminus \{3\}$
$\{1,2,3\} \cap \{6,7\} = \emptyset$	$\mathbb{Z} \setminus \{-1,-2,-3,\ldots\} = \mathbb{N}_0$
$(\{1,2,3\} \cap \{2,3,4\}) \setminus \{1,2,5\} = \{3\}$	$\{2,\pi,-3\} \cap \mathbb{N} = \{2\}$
$(\{1,2,3\} \setminus \{1,2\}) \setminus \{1\} = \{3\}$	$\{1,2,3\} \setminus (\{1,2\} \setminus \{1\}) = \{1,3\}$

Wir können die Vereinigung und den Durchschnitt von Mengen auf natürliche Weise auch auf mehr als zwei Mengen erweitern. Sind $A_1, \ldots, A_n$ Teilmengen einer Menge M, so definieren wir

$$\bigcup_{i=1}^{n} A_i = \{x \in M \mid \text{es gibt einen Index } i \in \{1,2,\ldots,n\} \text{ mit } x \in A_i\}$$

und

$$\bigcap_{i=1}^{n} A_i = \{x \in M \mid \text{für alle Indizes } i \in \{1,2,\ldots,n\} \text{ gilt } x \in A_i\}.$$

[1] Das Wort „oder“ wird in der Mathematik nicht im Sinne von „entweder ... oder“ verstanden, sondern immer als „das eine, das andere oder beides“.

Beispiele

- Es sei S die Menge aller Informatikstudenten in Bonn und es sei $S_i \subseteq S$ die Menge aller Informatikstudenten an der Universität Bonn im i-ten Semester. Dann gilt $S = \bigcup_{i=1}^{100} S_i$, da die Bonner Informatik 1969 (also vor weniger als 100 Semestern) gegründet wurde. Der Index, bei dem die Vereinigung startet, muss nicht immer gleich 1 sein. Beispielsweise können wir die Menge aller Studenten, deren Semesteranzahl zwischen 10 und 20 liegt, als $\bigcup_{i=10}^{20} S_i$ schreiben.
- Es sei S die Menge aller Teilnehmer dieser Vorlesung. Wir nehmen an, dass die Klausur am Ende des Semesters aus 10 Aufgaben besteht und wir bezeichnen mit $A_i \subseteq S$ die Studenten, die volle Punktzahl bei Aufgabe i erreichen. Dann besteht die Menge $\bigcap_{i=1}^{10} A_i$, aus allen Studenten, die insgesamt in der Klausur die volle Punktzahl erreichen.

Zwei Mengen A und B mit $A \cap B = \emptyset$ nennen wir *disjunkt*.

Eine Menge, die nur eine endliche Anzahl an Elementen enthält, nennen wir *endliche Menge*. Für eine endliche Menge M bezeichnen wir mit $|M|$ ihre *Kardinalität*, d. h. die Anzahl an Elementen, die sie enthält. Es gilt also beispielsweise $|\{5\}| = 1$ und $|\{2, 3, 5, 7\}| = 4$, wohingegen die Mengen $\mathbb{N}$ und $\mathbb{R}$ *unendliche Mengen* sind, auf die wir später noch genauer zu sprechen kommen. Man überlegt sich anschaulich anhand eines Venn-Diagramms leicht, dass für zwei beliebige endliche Mengen A und B stets die folgende Gleichung gilt:

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Eine weitere Möglichkeit, zwei Mengen A und B zu kombinieren, ist das *kartesische Produkt* $A \times B$. Die Elemente von $A \times B$ sind alle *geordneten Paare* (a, b) mit $a \in A$ und $b \in B$, also

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}.$$

Wir nennen diese Paare geordnet, da die Reihenfolge der Elemente wichtig ist und im Allgemeinen $(a, b) \neq (b, a)$ gilt. Man sollte das Paar (a, b) nicht mit der Menge $\{a, b\} = \{b, a\}$ verwechseln.

Beispiele

- Für $A = \{\text{links}, \text{rechts}\}$ und $B = \{\text{oben}, \text{unten}\}$ gilt

  $$A \times B = \{(\text{links}, \text{oben}), (\text{rechts}, \text{oben}), (\text{links}, \text{unten}), (\text{rechts}, \text{unten})\}.$$

- Für $A = \{a, b, c, d, e, f, g, h\}$ und $B = \{1, 2, 3, 4, 5, 6, 7, 8\}$ gilt

  $$A \times B = \{(a, 1), \ldots, (a, 8), (b, 1), \ldots, (b, 8), \ldots, (h, 1), \ldots, (h, 8)\}.$$

  Dies entspricht den Koordinaten eines Schachbretts.
- Die Menge $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$ enthält alle Koordinaten eines zweidimensionalen Koordinatensystems.

Das kartesische Produkt kann auch auf mehr als zwei Mengen verallgemeinert werden. Sind n Mengen $A_1, \ldots, A_n$ gegeben, so enthält die Menge $A_1 \times \ldots \times A_n$ alle *n-Tupel* $(a_1, \ldots, a_n)$ mit $a_i \in A_i$ für jedes i, also

$$A_1 \times \ldots \times A_n = \{(a_1, \ldots, a_n) \mid a_i \in A_i \text{ für alle } i \in \{1, \ldots, n\}\}.$$

Man überlegt sich leicht, dass für endliche Mengen $A_1, \ldots, A_n$ stets

$$|A_1 \times \ldots \times A_n| = |A_1| \cdot |A_2| \cdot \ldots \cdot |A_n|$$

gilt.

Zu guter Letzt betrachten wir in diesem Abschnitt noch den Begriff der *Potenzmenge*. Für eine Menge M bezeichnen wir mit $\mathcal{P}(M)$ ihre Potenzmenge. Dabei handelt es sich um die Menge aller Teilmengen von M, also

$$\mathcal{P}(M) = \{X \mid X \subseteq M\}.$$

In der Literatur wird die Potenzmenge oft statt mit $\mathcal{P}(M)$ auch mit 2^M bezeichnet.

Beispiele

- Für $M = \emptyset$ gilt $\mathcal{P}(M) = \{\emptyset\}$.
- Für $M = \{1\}$ gilt $\mathcal{P}(M) = \{\emptyset, \{1\}\}$.
- Für $M = \{1\}$ gilt $\mathcal{P}(\mathcal{P}(M)) = \{\emptyset, \{\emptyset\}, \{\{1\}\}, \{\emptyset, \{1\}\}\}$.
- Für $M = \{1, 2\}$ gilt $\mathcal{P}(M) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$.

Man beachte, dass $\mathcal{P}(M)$ eine Menge von Mengen ist. Für $M = \{1, 2\}$ gilt also nicht $1 \in \mathcal{P}(M)$, sondern $\{1\} \in \mathcal{P}(M)$, denn nicht das Objekt 1 gehört zur Potenzmenge, sondern die Menge, die aus dem Objekt 1 besteht. Ebenso gilt zum Beispiel $\{1, 2\} \in \mathcal{P}(M)$. Außerdem gilt $\emptyset \in \mathcal{P}(M)$ und $M \in \mathcal{P}(M)$ für jede Menge M. Für eine Menge X sind die Aussagen $X \in \mathcal{P}(M)$ und $X \subseteq M$ äquivalent.

Beispiel

Wir betrachten einen einfachen Getränkeautomaten, der Orangensaft und Apfelsaft verkauft. Von jedem Saft hat er maximal 50 Packungen auf Vorrat. Außerdem besitzt er eine gelbe Warnleuchte, die anzeigt, dass er kein Wechselgeld mehr herausgeben kann, und eine rote Warnleuchte, die anzeigt, dass er keine Scheine mehr akzeptiert. Diese Warnleuchten können unabhängig voneinander aus- oder eingeschaltet sein. Die Menge der möglichen Zustände dieses Automaten können wir als

$$\mathcal{P}(\{\text{gelb}, \text{rot}\}) \times \{0, 1, 2, \ldots, 50\} \times \{0, 1, 2, \ldots, 50\}$$

beschreiben. Ein Element dieser Menge ist zum Beispiel $(\emptyset, 30, 20)$. Es entspricht dem Zustand, in dem beide Warnleuchten ausgeschaltet sind und noch 30 Packungen Orangensaft sowie 20 Packungen Apfelsaft vorrätig sind. Sind bei diesem Vorrat die Warnleuchten beide eingeschaltet, so befindet sich der Automat im Zustand $(\{\text{gelb}, \text{rot}\}, 30, 20)$.

2.2 Beweise

Bei einem mathematischen Beweis geht es darum, die Richtigkeit einer *Aussage* nachzuweisen. Dabei startet man mit einer Menge von Grundaussagen, die man als wahr voraussetzt. Dies sind die sogenannten *Axiome*. Ein Axiom, auf dem die ganze Mathematik aufbaut, ist zum Beispiel, dass jede natürliche Zahl einen Nachfolger besitzt. Bei einem *direkten Beweis* wird aus diesen Axiomen durch logische Schlussfolgerungen die Richtigkeit weiterer Aussagen nachgewiesen, solange bis man bei der zu beweisenden Aussage angelangt ist. Bei einem *indirekten Beweis* geht man davon aus, dass die zu beweisende Aussage falsch ist und leitet aus dieser Annahme einen Widerspruch her.

2.2.1 Aussagen

Unter einer *Aussage* verstehen wir zunächst ganz allgemein Sätze, die *wahr* oder *falsch* sein können. Im Gegensatz zu alltäglichen Aussagen wie „Bonn ist eine schöne Stadt“, über die man geteilter Meinung sein kann, interessieren wir uns für Aussagen, die wie die folgenden Beispiele einen eindeutigen *Wahrheitswert* besitzen.

- 11 ist eine Primzahl. (Diese Aussage ist wahr.)
- 15 ist die Summe von 8 und 5. (Diese Aussage ist falsch.)
- Jede gerade Zahl größer als 2 kann als Summe zweier Primzahlen geschrieben werden. (Diese Aussage besitzt einen eindeutigen Wahrheitswert, der aber bis heute unbekannt ist. Es handelt sich um die Goldbachsche Vermutung.)

Aussagen bezeichnen wir meistens mit großen Buchstaben $A, B, C, \ldots$ und für eine Aussage A bezeichnen wir mit $w(A)$ ihren Wahrheitswert. Dabei bedeutet $w(A) = 0$, dass die Aussage A falsch ist, und $w(A) = 1$ bedeutet, dass sie wahr ist. Es gilt also

$$w(\text{„11 ist eine Primzahl“}) = 1 \quad \text{und} \quad w(\text{„15 ist eine Primzahl“}) = 0.$$

Ist A eine wahre Aussage, so sagen wir, dass A *gilt*. Ist A hingegen eine falsche Aussage, so sagen wir, dass A *nicht gilt*.

Aussagen werden erst dadurch interessant, dass man sie verknüpfen kann. Sind A und B zwei Aussagen, so ist auch $A \wedge B$ eine Aussage, die wir die *Konjunktion* von A und B nennen. Sie ist genau dann wahr, wenn beide Aussagen A und B wahr sind, und entspricht somit der Aussage „es gelten A und B.“ Auch $A \vee B$ ist eine Aussage, die wir die *Disjunktion* von A und B nennen. Sie ist genau dann wahr, wenn mindestens eine der Aussagen A und B wahr ist. Damit entspricht $A \vee B$ der Aussage „es gilt A oder B (oder beide).“ Für eine Aussage A bezeichnen wir mit $\neg A$ ihre *Negation*. Diese Aussage ist genau dann wahr, wenn A falsch ist. Damit entspricht sie der Aussage „A gilt nicht.“

Eine Verknüpfung können wir auch durch ihre *Wahrheitstabelle* beschreiben. In einer solchen Tabelle ist für jede mögliche Kombination von Wahrheitswerten der Wahrheitswert der Verknüpfung dargestellt. Die Wahrheitstabellen der drei oben definierten Verknüpfungen sehen wie folgt aus.

A	$\neg A$
0	1
1	0

A	B	$A \wedge B$
0	0	0
1	0	0
0	1	0
1	1	1

A	B	$A \vee B$
0	0	0
1	0	1
0	1	1
1	1	1

Zur weiteren Veranschaulichung betrachten wir drei Beispiele für Aussagen, die durch Verknüpfungen entstehen:

$$w(\text{„11 ist eine Primzahl“} \vee \text{„15 ist eine Primzahl“}) = 1,$$
$$w(\text{„11 ist eine Primzahl“} \wedge \text{„15 ist eine Primzahl“}) = 0,$$
$$\text{und } w(\neg\text{„11 ist eine Primzahl“}) = 0$$

Natürlich kann man Verknüpfungen auch auf Aussagen anwenden, die selbst durch Verknüpfungen entstanden sind. Sind A, B und C drei Aussagen, so ist beispielsweise auch $A \wedge ((\neg B) \vee C)$ eine Aussage. Durch die Klammern wird die Reihenfolge festgelegt, in der die Verknüpfungen angewendet werden. Auch für diese Aussage können wir wieder eine Wahrheitstabelle angeben. Die folgende Tabelle enthält noch zwei zusätzliche Spalten, in denen die Wahrheitswerte zweier Teilaussagen stehen. Diese Spalten helfen uns dabei, die Wahrheitswerte der eigentlichen Aussage zu bestimmen.

A	B	C	$\neg B$	$(\neg B) \vee C$	$A \wedge ((\neg B) \vee C)$
0	0	0	1	1	0
1	0	0	1	1	1
0	1	0	0	0	0
1	1	0	0	0	0
0	0	1	1	1	0
1	0	1	1	1	1
0	1	1	0	1	0
1	1	1	0	1	1

Um Klammern zu sparen, vereinbaren wir, dass die Verknüpfung $\neg$ vor den Verknüpfungen $\vee$ und $\wedge$ ausgewertet wird. Mit dieser Konvention vereinfacht sich der obige Term zu $A \wedge (\neg B \vee C)$.

2.2.2 Implikationen und Äquivalenzen

Um aus der Richtigkeit einer Aussage die Richtigkeit einer anderen Aussage abzuleiten, sind *Implikationen* und *Äquivalenzen* von Bedeutung. Sind A und B zwei Aussagen, so sind auch $A \Rightarrow B$ („A impliziert B“ oder „aus A folgt B“) und $A \iff B$ („A und B sind äquivalent“ oder „A gilt genau dann, wenn B gilt“) Aussagen. Formal sind diese beiden Aussagen über die folgenden Wahrheitstabellen definiert.

A	B	$A \Rightarrow B$
0	0	1
1	0	0
0	1	1
1	1	1

A	B	$A \iff B$
0	0	1
1	0	0
0	1	0
1	1	1

Sind die Aussagen A und $A \Rightarrow B$ wahr, dann befinden wir uns in der letzten Zeile der linken obigen Wahrheitstabelle. Es folgt also, dass auch die Aussage B wahr ist. Ist die Aussage A jedoch falsch und die Aussage $A \Rightarrow B$ wahr, so befinden wir uns entweder in der ersten oder der dritten Zeile der obigen Wahrheitstabelle. Ein Rückschluss auf den Wahrheitswert der Aussage B ist also nicht möglich. Die Bedeutung der Aussage $A \Rightarrow B$ können wir wie folgt zusammenfassen: Wenn die Aussage A wahr ist, dann ist auch die Aussage B wahr. Ist die Aussage A hingegen falsch, so liefert die Implikation $A \Rightarrow B$ keine Information über den Wahrheitswert von Aussage B.

Ist die Aussage $A \iff B$ wahr, so ist die Beziehung zwischen den Aussagen A und B noch enger. Gilt $A \iff B$ und ist die Aussage A wahr, so befinden wir uns in der letzten Zeile der rechten obigen Wahrheitstabelle. Demnach ist also auch die Aussage B wahr. Gilt $A \iff B$ und ist die Aussage A falsch, so befinden wir uns in der ersten Zeile der Wahrheitstabelle. Dann ist die Aussage B also falsch. Die Bedeutung der Aussage $A \iff B$ können wir wie folgt zusammenfassen: Die Aussage A ist genau dann wahr, wenn die Aussage B wahr ist.

Es ist wichtig zu verstehen, dass aus der Gültigkeit der Aussagen $A \Rightarrow B$ und $A \iff B$ allein keine Rückschlüsse auf die Wahrheitswerte von A und B gezogen werden können. Die Wahrheitswerte von A und B werden dadurch lediglich zueinander in Beziehung gesetzt.

Beispiele

- $x \in \mathbb{N} \Rightarrow x \in \mathbb{Z}$ (wahr für jedes $x \in \mathbb{R}$)
- $x \in \mathbb{Q} \Rightarrow x \in \mathbb{N}$ (im Allgemeinen falsch, z. B. für $x = 3.5$)
- $x \in \mathbb{Z} \Rightarrow x \in \mathbb{N}_0$ (im Allgemeinen falsch, z. B. für $x = -1$)
- $x \in \mathbb{Z} \iff (x \in \mathbb{N}_0 \vee -x \in \mathbb{N}_0)$ (wahr für jedes $x \in \mathbb{R}$)
- $(x = x + 1) \Rightarrow (x = x)$ (wahr für jedes $x \in \mathbb{R}$)
- $(x = x) \Rightarrow (x = x + 1)$ (falsch für jedes $x \in \mathbb{R}$)
- $(x = x + 1) \iff (x = x - 1)$ (wahr für jedes $x \in \mathbb{R}$)

Um Klammern zu sparen, vereinbaren wir, dass die Verknüpfungen $\wedge$, $\vee$ und $\neg$ vor den Verknüpfungen $\iff$ und $\Rightarrow$ ausgewertet werden. Damit vereinfacht sich beispielsweise der Term $A \iff (\neg B \vee C)$ zu $A \iff \neg B \vee C$.

Das folgende Theorem fasst einige wichtige Möglichkeiten zusammen, wie wir vom Wahrheitswert einer Aussage auf den Wahrheitswert einer anderen Aussage schließen können.

Theorem 2.1. *Es seien A, B und C drei beliebige Aussagen.*

a) Die Aussage $A \iff B$ gilt genau dann, wenn $A \Rightarrow B$ und $B \Rightarrow A$ gelten.

b) Ist die Aussage A wahr und gilt $A \Rightarrow B$, so ist auch die Aussage B wahr.

c) Ist die Aussage B falsch und gilt $A \Rightarrow B$, so ist auch die Aussage A falsch.

d) Gilt $A \Rightarrow B$ und $B \Rightarrow C$, so gilt auch $A \Rightarrow C$.

e) Gilt $A \iff B$ *und* $B \iff C$*, so gilt auch* $A \iff C$*.*

f) Die Implikationen $A \Rightarrow B$ *und* $\neg B \Rightarrow \neg A$ *sind äquivalent.*

g) Gilt $A \Rightarrow B$*,* $B \Rightarrow C$ *und* $C \Rightarrow A$*, so sind die Aussagen* A*,* B *und* C *äquivalent.*

Beweis. Wir führen exemplarisch den Beweis der ersten Teilaussage mithilfe der folgenden Wahrheitstabelle.

A	B	$A \Rightarrow B$	$B \Rightarrow A$	$(A \Rightarrow B) \wedge (B \Rightarrow A)$	$A \iff B$
0	0	1	1	1	1
1	0	0	1	0	0
0	1	1	0	0	0
1	1	1	1	1	1

Wir stellen fest, dass die letzten beiden Spalten übereinstimmen, was zu beweisen war. Auch die weiteren Teilaussagen können mit Hilfe von entsprechenden Wahrheitstabellen bewiesen werden. Dies überlassen wir dem Leser als Übung. □

Die folgenden beiden *De Morgan'schen Gesetze* sind oft hilfreich bei der Umformung von Aussagen.

Theorem 2.2. *Es seien* A *und* B *beliebige Aussagen. Es gilt*

a) $\neg(A \wedge B) \iff \neg A \vee \neg B$*;*

b) $\neg(A \vee B) \iff \neg A \wedge \neg B$*.*

Ebenso wie Theorem 2.1 kann auch dieses Theorem durch eine einfache Betrachtung der Wahrheitstabellen bewiesen werden. Wir überlassen den Beweis dem Leser als Übung.

2.2.3 Direkte und indirekte Beweise

Wir unterscheiden zwischen *direkten* und *indirekten Beweisen*. Bei einem direkten Beweis wird die zu beweisende Aussage aus den Axiomen hergeleitet. Es wird also gezeigt, dass die Axiome, die per Definition als wahr vorausgesetzt werden, die zu beweisende Aussage implizieren. Bei einem indirekten Beweis (auch *Widerspruchsbeweis* genannt) geht man davon aus, dass die zu beweisende Aussage falsch ist, und leitet aus dieser Annahme einen Widerspruch her. Es ist allerdings nicht praktikabel, jede Aussage komplett auf die Axiome der Mathematik zurückzuführen. Im Folgenden nehmen wir deshalb viele Aussagen, die zum Schulwissen gehören, als bereits bewiesen an.

Theorem 2.3. *Das Quadrat jeder ungeraden natürlichen Zahl ist ungerade.*

Beweis. Es sei $n \in \mathbb{N}$ eine beliebige ungerade natürliche Zahl. Wir setzen $k = (n-1)/2$. Da $n-1$ eine gerade Zahl ist, gilt $k \in \mathbb{N}_0$. Außerdem gilt $n = 2k+1$. Es folgt

$$n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1.$$

Die Zahl $z = 2(2k^2 + 2k)$ ist ein Vielfaches von 2 und damit gerade. Daraus folgt, dass die Zahl $n^2 = z + 1$ ungerade ist □

Schauen wir uns den vorangegangenen Beweis an, so stellen wir fest, dass dort weder das Zeichen $\Rightarrow$ noch das Zeichen $\Longleftrightarrow$ auftaucht. Um den Beweis lesbarer zu gestalten, haben wir alle Implikationen in Worten beschrieben. Formal entspricht der Beweis der folgenden Implikationskette, in der wir k abkürzend für $\frac{n-1}{2}$ verwenden.

$$\begin{aligned} & n \text{ ist ungerade} \\ \Rightarrow\ & k \in \mathbb{N}_0 \text{ und } n = 2k+1 \\ \Rightarrow\ & 2k^2+2k \in \mathbb{N}_0 \text{ und } n^2 = 2(2k^2+2k)+1 \\ \Rightarrow\ & n^2 \text{ ist ungerade} \end{aligned}$$

Nun geben wir ein Beispiel für einen indirekten Beweis.

Theorem 2.4. *Es gilt* $\sqrt{2} \notin \mathbb{Q}$.

Beweis. Wir wollen zeigen, dass es keine ganzen Zahlen $a \in \mathbb{Z}$ und $b \in \mathbb{Z} \setminus \{0\}$ gibt, für die $\frac{a}{b} = \sqrt{2}$ gilt. Wir führen einen Widerspruchsbeweis und gehen davon aus, dass es zwei solche Zahlen a und b gibt. Der größte gemeinsame Teiler von a und b kann zunächst beliebig groß sein. Wir können aber beide Zahlen durch ihren größten gemeinsamen Teiler dividieren und erhalten dann zwei teilerfremde[1] Zahlen $a' \in \mathbb{Z}$ und $b' \in \mathbb{Z} \setminus \{0\}$ mit $\frac{a'}{b'} = \sqrt{2}$. Das bedeutet, wir können *ohne Beschränkung der Allgemeinheit (o. B. d. A.)* direkt davon ausgehen, dass die Zahlen a und b teilerfremd sind. Wegen $\sqrt{2} > 0$ können wir ebenfalls o. B. d. A. davon ausgehen, dass sowohl a als auch b positiv sind.

Es gilt

$$\frac{a}{b} = \sqrt{2} \ \Rightarrow\ \frac{a^2}{b^2} = 2 \ \Rightarrow\ a^2 = 2b^2.$$

Damit ist a^2 eine gerade Zahl. Mit dem folgenden Lemma[2], das wir weiter unten beweisen, folgt daraus, dass auch die Zahl a gerade ist.

Lemma 2.5. *Die Wurzel aus einer geraden Quadratzahl*[3] *ist gerade.*

Da a eine gerade Zahl ist, gibt es ein $k \in \mathbb{N}$ mit $a = 2k$. Mit der obigen Gleichung ergibt sich daraus

$$a^2 = 2b^2 \ \Rightarrow\ (2k)^2 = 2b^2 \ \Rightarrow\ b^2 = 2k^2.$$

Somit ist auch b^2 eine gerade Zahl. Wieder können wir Lemma 2.5 anwenden und erhalten, dass b eine gerade Zahl ist. Demzufolge ist 2 ein gemeinsamer Teiler von a und b. Dies ist ein Widerspruch zu der Annahme, dass a und b teilerfremd sind.

Zusammenfassend haben wir die folgende Implikation gezeigt:

$$\sqrt{2} \in \mathbb{Q} \ \Rightarrow\ \text{es gibt zwei gerade teilerfremde Zahlen } a \text{ und } b.$$

[1] Zwei Zahlen heißen *teilerfremd*, wenn ihr größter gemeinsamer Teiler 1 ist.
[2] Mit *Lemma* bezeichnet man in der Mathematik einen Hilfssatz.
[3] Eine *Quadratzahl* ist das Quadrat einer natürlichen Zahl, also $1 = 1^2, 4 = 2^2, 9 = 3^2, \ldots$.

Wäre die Aussage auf der linken Seite wahr, so würde mit der Implikation folgen, dass auch die Aussage auf der rechten Seite wahr ist. Da diese aber offensichtlich falsch ist, muss auch die Aussage auf der linken Seite falsch sein. Dies entspricht Theorem 2.1 c). Wir haben also abgesehen von dem noch ausstehenden Beweis des obigen Lemmas bewiesen, dass $\sqrt{2} \notin \mathbb{Q}$ gilt. □

Beweis von Lemma 2.5. Das Lemma lässt sich mit einem einfachen Widerspruchsbeweis auf Theorem 2.3 zurückführen. Es sei $n \in \mathbb{N}$ eine beliebige gerade Quadratzahl. Dann gibt es ein $k \in \mathbb{N}$ mit $k^2 = n$. Wir nehmen an, dass k ungerade ist. Mit Theorem 2.3 folgt, dass dann auch $k^2 = n$ ungerade ist. Dies ist ein Widerspruch zu der Annahme, dass n gerade ist. □

Als letztes Beispiel präsentieren wir noch den klassischen Widerspruchsbeweis von Euklid, dass es unendlich viele Primzahlen gibt.

Theorem 2.6. *Es gibt unendlich viele Primzahlen.*

Beweis. Wir gehen davon aus, dass es nur endlich viele Primzahlen gibt. Diese nennen wir $p_1, p_2, \ldots, p_k$. Nun betrachten wir die Zahl $m = p_1 \cdot \ldots \cdot p_k + 1$. Diese ist größer als alle Primzahlen $p_1, \ldots, p_k$. Wäre m also selbst eine Primzahl, so hätten wir eine neue Primzahl gefunden. Dies steht im Widerspruch zu der Annahme, dass $p_1, \ldots, p_k$ eine vollständige Liste aller Primzahlen ist.

Es bleibt also nur noch die Möglichkeit, dass m keine Primzahl ist. Dann gibt es eine Primzahl q, die die Zahl m teilt[4]. Keine der Zahlen p_i teilt jedoch m, da m bei Division durch ein p_i stets den Rest 1 lässt. Also ist q verschieden von allen Primzahlen $p_1, \ldots, p_k$. Damit haben wir eine neue Primzahl gefunden und erhalten erneut einen Widerspruch zu der Annahme, dass $p_1, \ldots, p_k$ eine vollständige Liste aller Primzahlen ist. Der Beweis ist abgeschlossen, da wir in beiden möglichen Fällen einen Widerspruch erhalten haben. □

2.2.4 Vollständige Induktion

Oft treffen wir auf Aussagen, die mit einer natürlichen Zahl parametrisiert sind. Beispielsweise können wir für jedes $n \in \mathbb{N}_0$ die Aussage $A(n)$ formulieren, dass die Potenzmenge $\mathcal{P}(M)$ jeder Menge M mit n Elementen die Kardinalität 2^n besitzt. Hat man eine solche Folge von Aussagen $A(0), A(1), A(2), \ldots$, so kann man oft das Prinzip der *vollständigen Induktion* anwenden, um die Richtigkeit all dieser Aussagen auf einmal nachzuweisen. Dazu sind die folgenden beiden Schritte nötig.

1. Beim *Induktionsanfang* weist man nach, dass die erste Aussage $A(k)$ gilt, wobei k je nach Nummerierung der Aussagen variieren kann. In dem obigen Beispiel würden wir $k = 0$ setzen.

[4]Dies ist ein Beispiel für eine Aussage, die wir mit Schulwissen als wahr voraussetzen, ohne sie auf die Axiome zurückzuführen.

2. Im *Induktionsschritt* weist man nach, dass $A(n) \Rightarrow A(n+1)$ für jedes $n \geq k$ gilt. Die Aussage $A(n)$ nennt man die *Induktionsvoraussetzung*.

Hat man diese beiden Schritte durchgeführt, so hat man bewiesen, dass alle Aussagen $A(k), A(k+1), A(k+2), \ldots$ gelten. Die Richtigkeit von $A(k)$ wird explizit im Induktionsanfang bewiesen. Der Induktionsschritt besagt unter anderem, dass $A(k) \Rightarrow A(k+1)$ gilt. Mit Theorem 2.1 b) folgt also direkt, dass auch $A(k+1)$ gilt. Dieses Vorgehen können wir iterieren, denn der Induktionsschritt besagt auch, dass $A(k+1) \Rightarrow A(k+2)$ gilt. Demzufolge folgt mit Theorem 2.1 b), dass $A(k+2)$ gilt usw.

Notationen

Es seien $a_1, a_2, \ldots, a_n \in \mathbb{R}$ reelle Zahlen und es seien $j, k \in \mathbb{N}$ mit $1 \leq j \leq k \leq n$. Wir benutzen häufig die Notationen

$$\sum_{i=j}^{k} a_i = a_j + a_{j+1} + \ldots + a_k$$

und

$$\prod_{i=j}^{k} a_i = a_j \cdot a_{j+1} \cdot \ldots \cdot a_k.$$

Für $j > k$ definieren wir außerdem

$$\sum_{i=j}^{k} a_i = 0 \quad \text{und} \quad \prod_{i=j}^{k} a_i = 1.$$

Wir betrachten ein Beispiel für eine Aussage, die mit vollständiger Induktion bewiesen werden kann.

Theorem 2.7. *Für jedes $n \in \mathbb{N}$ gilt*

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Beweis. Für $n \in \mathbb{N}$ bezeichnen wir mit $A(n)$ die Aussage, dass die Gleichung

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

gilt. Damit erhalten wir eine Folge $A(1), A(2), A(3), \ldots$ von Aussagen.

Im Induktionsanfang zeigen wir die Richtigkeit der Aussage $A(1)$. Diese entspricht der Gleichung

$$\sum_{i=1}^{1} i = \frac{1(1+1)}{2}$$

und ist dementsprechend wahr, da beide Seiten gleich 1 sind.

Es sei nun $n \in \mathbb{N}$ beliebig. Im Induktionsschritt setzen wir die Induktionsvoraussetzung $A(n)$ als wahr voraus. Es gilt also die Gleichung

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Unsere Aufgabe ist es, die Richtigkeit der Aussage $A(n+1)$ zu zeigen. Es gilt

$$\begin{aligned}\sum_{i=1}^{n+1} i &= \left(\sum_{i=1}^{n} i\right) + (n+1) \overset{A(n)}{=} \frac{n(n+1)}{2} + (n+1) \\ &= \frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2}.\end{aligned}$$

In der ersten Zeile dieser Rechnung deuten wir durch $A(n)$ über dem Gleichheitszeichen an, dass die entsprechende Gleichung aus der Aussage $A(n)$ folgt. Die obige Gleichungskette entspricht genau der Aussage $A(n+1)$. Somit ist der Induktionsschritt abgeschlossen, da wir für jedes $n \in \mathbb{N}$ die Implikation $A(n) \Rightarrow A(n+1)$ gezeigt haben. □

Wir können das Prinzip der vollständigen Induktion auch anwenden, um die obige Aussage über die Kardinalität von Potenzmengen zu beweisen.

Theorem 2.8. *Für jedes $n \in \mathbb{N}_0$ und jede Menge M mit $|M| = n$ gilt $|\mathcal{P}(M)| = 2^n$.*

Beweis. Für den Induktionsanfang betrachten wir den Fall $n = 0$. Aus $|M| = 0$ folgt $M = \emptyset$. Dementsprechend gilt $\mathcal{P}(M) = \{\emptyset\}$, also $|\mathcal{P}(M)| = 1 = 2^0$. Somit ist der Induktionsanfang gezeigt.

Für den Induktionsschritt sei $n \in \mathbb{N}_0$ beliebig. Wir nehmen an, dass $|\mathcal{P}(M)| = 2^n$ für jede Menge M mit $|M| = n$ gilt. Es sei nun M eine beliebige Menge mit $|M| = n+1$ Elementen und es sei $x \in M$ ein beliebiges Element von M. Wir teilen die Potenzmenge $\mathcal{P}(M)$ in zwei disjunkte Mengen $\mathcal{P}_1$ und $\mathcal{P}_2$ ein:

$$\mathcal{P}_1 = \{Y \in \mathcal{P}(M) \mid x \notin Y\} \quad \text{und} \quad \mathcal{P}_2 = \{Y \in \mathcal{P}(M) \mid x \in Y\}.$$

Es gilt dann $\mathcal{P}(M) = \mathcal{P}_1 \cup \mathcal{P}_2$ und $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$, also $|\mathcal{P}(M)| = |\mathcal{P}_1| + |\mathcal{P}_2|$.

Beispiel

Sei $M = \{1, 2, 3\}$ und $x = 3$. Dann gilt

$$\mathcal{P}(M) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

sowie

$$\mathcal{P}_1 = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} \quad \text{und} \quad \mathcal{P}_2 = \{\{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

Die Menge $\mathcal{P}_1$ enthält alle Teilmengen von M, die das Element x nicht enthalten. Es gilt also $\mathcal{P}_1 = \mathcal{P}(M \setminus \{x\})$. Die Menge $M \setminus \{x\}$ enthält n Elemente, weshalb wir die

Induktionsvoraussetzung anwenden können. Diese besagt, dass die Potenzmenge jeder Menge mit n Elementen die Kardinalität 2^n besitzt. Also gilt $|\mathcal{P}_1| = |\mathcal{P}(M \setminus \{x\})| = 2^n$.

Als nächstes betrachten wir die Menge $\mathcal{P}_2$, die aus allen Teilmengen von M besteht, die das Element x enthalten. Entfernen wir aus jeder Menge in $\mathcal{P}_2$ das Element x, so erhalten wir wieder die Potenzmenge von $M \setminus \{x\}$ (siehe obiges Beispiel). Formal gilt

$$\mathcal{P}_2 = \{Y \subseteq M \mid Y = X \cup \{x\} \text{ für ein } X \in \mathcal{P}(M \setminus \{x\})\}.$$

Da wir die Elemente von $\mathcal{P}_2$ und die Elemente von $\mathcal{P}(M \setminus \{x\})$ auf diese Weise eindeutig zueinander in Beziehung setzen können, haben diese beiden Mengen die gleiche Kardinalität. Wir werden uns später noch ausführlicher mit der Kardinalität von Mengen beschäftigen, nehmen $|\mathcal{P}_2| = |\mathcal{P}(M \setminus \{x\})|$ hier aber erst mal als bewiesen an. Nun können wir wieder die Induktionsvoraussetzung anwenden und erhalten $|\mathcal{P}_2| = |\mathcal{P}(M \setminus \{x\})| = 2^n$.

Insgesamt ergibt sich

$$|\mathcal{P}(M)| = |\mathcal{P}_1| + |\mathcal{P}_2| = 2^n + 2^n = 2^{n+1}.$$

Damit ist der Induktionsschritt abgeschlossen und das Theorem ist bewiesen. □

2.3 Quantoren

Oft treten Aussagen auf, in denen Variablen vorkommen. Benutzen wir n als Variable für eine natürliche Zahl, so können wir beispielsweise die Aussage „n ist eine Quadratzahl“ formulieren. Diese Aussage kürzen wir im Folgenden mit $A(n)$ ab. Der Wahrheitswert einer solchen Aussage hängt im Allgemeinen natürlich davon ab, welchen Wert die Variable n annimmt. Anstatt der Variablen einen konkreten Wert zuzuweisen, interessiert uns häufig, ob es mindestens eine natürliche Zahl n gibt, für die die Aussage $A(n)$ wahr ist, oder ob sie sogar für jede natürliche Zahl n wahr ist.

Um dies zu formalisieren, führen wir *Quantoren* ein. Mit dem *Existenzquantor* $\exists$ können wir die Aussage $\exists n \in \mathbb{N} : A(n)$ formulieren. Diese ist genau dann wahr, wenn es mindestens eine natürliche Zahl n gibt, für die die Aussage $A(n)$ wahr ist. Für das obige Beispiel ist die Aussage $\exists n \in \mathbb{N} : A(n)$ wahr, denn die Zahl $n = 1$ ist eine Quadratzahl. Mit dem *Allquantor* $\forall$ können wir die Aussage $\forall n \in \mathbb{N} : A(n)$ formulieren. Diese ist genau dann wahr, wenn die Aussage $A(n)$ für jede natürliche Zahl n wahr ist. Für das obige Beispiel ist die Aussage $\forall n \in \mathbb{N} : A(n)$ falsch, da beispielsweise $n = 2$ keine Quadratzahl ist.

Um den Umgang mit diesen Quantoren zu üben, betrachten wir noch einige Beispiele. Die Aussage $A(n)$, die wir oben als Beispiel gegeben haben, können wir selbst mit einem Quantor darstellen:

$$A(n) \iff \exists k \in \mathbb{N} : n = k^2.$$

Als nächstes wollen wir die Aussage $P(n)$, dass die Zahl n eine Primzahl ist, formulieren. Dazu geben wir zunächst die Negation von $P(n)$ an, die besagt, dass die Zahl n keine Primzahl ist:

$$\neg P(n) \iff (n = 1) \vee (\exists x \in \mathbb{N} : \exists y \in \mathbb{N} : (x \geq 2) \wedge (y \geq 2) \wedge (n = x \cdot y)).$$

Die Aussage $P(n)$ kann nun einfach durch die Negation dieser Aussage ausgedrückt werden:

$$P(n) \iff \neg((n = 1) \vee (\exists x \in \mathbb{N} : \exists y \in \mathbb{N} : (x \geq 2) \wedge (y \geq 2) \wedge (n = x \cdot y))).$$

Jede natürliche Zahl ist entweder gleich 1, eine Primzahl oder sie besitzt einen echten Teiler. Die folgende Aussage ist also wahr

$$\forall n \in \mathbb{N} : (n = 1) \vee P(n) \vee (\exists x \in \mathbb{N} : (x > 1) \wedge (x < n) \wedge (x \text{ teilt } n)).$$

Auch die Goldbachsche Vermutung, dass jede gerade Zahl größer als 2 als Summe zweier Primzahlen geschrieben werden kann, können wir mithilfe von Quantoren schreiben. Auf den ersten Blick scheint es problematisch zu sein, dass der Allquantor eine Aussage über alle natürlichen Zahlen macht und nicht nur über die geraden Zahlen größer als 2. Dieses Problem können wir aber lösen und die Goldbachsche Vermutung wie folgt mithilfe von Quantoren ausdrücken:

$$\forall n \in \mathbb{N} : (n \leq 2) \vee (n \text{ ist ungerade}) \vee (\exists x \in \mathbb{N} : \exists y \in \mathbb{N} : P(x) \wedge P(y) \wedge (n = x + y)).$$

In den vorangegangenen Beispielen beziehen sich die Quantoren stets auf die Menge der natürlichen Zahlen. Später werden wir Quantoren auch für andere Objekte wie zum Beispiel reelle Zahlen einsetzen. Den Wertebereich, auf den die Quantoren sich beziehen, werden wir stets explizit angeben, wenn er nicht eindeutig aus dem Kontext hervorgeht.

Ein verbreiteter Fehler bei Widerspruchsbeweisen ist die inkorrekte Negation von Aussagen. In diesem Zusammenhang sei noch einmal an die De Morgan'schen Gesetze in Theorem 2.2 erinnert. Für Aussagen, in denen Quantoren vorkommen, ist das folgende Theorem wichtig. Wir werden es nicht formal beweisen, der Leser sollte sich aber intuitiv von seiner Richtigkeit überzeugen.

Theorem 2.9. *Es sei M eine beliebige Menge und für jedes $x \in M$ sei $A(x)$ eine Aussage. Es gilt*

a) $\neg(\exists x \in M : A(x)) \iff \forall x \in M : \neg A(x)$ *und*
b) $\neg(\forall x \in M : A(x)) \iff \exists x \in M : \neg A(x)$.

Wir greifen noch einmal die Aussage $P(n)$ von oben auf. Es gilt

$$\begin{aligned} P(n) &\iff \neg((n = 1) \vee (\exists x \in \mathbb{N} : \exists y \in \mathbb{N} : (x \geq 2) \wedge (y \geq 2) \wedge (n = x \cdot y))) \\ &\iff (n \neq 1) \wedge (\forall x \in \mathbb{N} : \neg(\exists y \in \mathbb{N} : (x \geq 2) \wedge (y \geq 2) \wedge (n = x \cdot y))) \\ &\iff (n \neq 1) \wedge (\forall x \in \mathbb{N} : \forall y \in \mathbb{N} : \neg((x \geq 2) \wedge (y \geq 2) \wedge (n = x \cdot y))) \end{aligned}$$

$$\iff (n \neq 1) \wedge (\forall x \in \mathbb{N} : \forall y \in \mathbb{N} : \neg(x \geq 2) \vee \neg(y \geq 2) \vee \neg(n = x \cdot y))$$
$$\iff (n \neq 1) \wedge (\forall x \in \mathbb{N} : \forall y \in \mathbb{N} : (x = 1) \vee (y = 1) \vee (n \neq x \cdot y)).$$

Quantoren können sich auch auf die leere Menge beziehen. Es sei M eine beliebige Menge und für jedes $x \in M$ sei $A(x)$ eine Aussage. Dann definieren wir die Aussage $\forall x \in \emptyset : A(x)$ als wahr und die Aussage $\exists x \in \emptyset : A(x)$ als falsch. Dies ist also vollkommen unabhängig von den Aussagen $A(x)$.

2.4 Relationen und Abbildungen

Oft setzt man die Elemente verschiedener Mengen zueinander in Beziehung. Durch die Zuweisung der Übungsgruppen werden beispielsweise die Menge der Hörer dieser Vorlesung und die Menge der angebotenen Übungsgruppen zueinander in Beziehung gesetzt. Um solche Beziehungen mathematisch beschreiben zu können, führen wir in diesem Abschnitt *Relationen* und *Abbildungen* ein.

2.4.1 Relationen

Definition 2.10. *Es seien M und N zwei beliebige Mengen. Eine Teilmenge $R \subseteq M \times N$ des kartesischen Produktes von M und N bezeichnen wir als* (binäre) Relation *zwischen M und N. Wir sagen, dass $a \in M$ zu $b \in N$* in Relation R steht*, wenn $(a, b) \in R$ gilt. Statt $(a, b) \in R$ und $(a, b) \notin R$ schreiben wir auch $a\,R\,b$ bzw. $a \not R\, b$. Gilt $M = N$ so sagen wir, dass R eine* Relation auf der Menge M *ist.*

*Es seien $M_1, M_2, \ldots, M_n$ beliebige Mengen. Eine n-*stellige Relation *zwischen diesen Mengen ist eine Teilmenge des kartesischen Produktes $M_1 \times M_2 \times \ldots \times M_n$.*

Beispiele

- Ist S die Menge aller Hörer dieser Vorlesung und ist U die Menge aller Übungsgruppen, so bildet
$$R = \{(s, u) \in S \times U \mid \text{Student } s \text{ besucht Übungsgruppe } u\}$$
eine Relation zwischen S und U.
- Es gibt einige uns wohlbekannte Relationen auf der Menge der reellen Zahlen wie z. B. $<$. Dabei steht $a \in \mathbb{R}$ genau dann zu $b \in \mathbb{R}$ in Relation $<$, wenn a kleiner als b ist. Auch $R = \{(x, y) \in \mathbb{R}^2 \mid x - y = 1\}$ ist eine Relation auf $\mathbb{R}$. Für diese gilt zum Beispiel $3\,R\,2$ und $1{,}2\,R\,0{,}2$, aber $1{,}1 \not R\, 2{,}3$.

- Es sei $n \in \mathbb{N}$ beliebig. Eine wichtige Relation auf $\mathbb{Z}$, auf die wir später noch ausführlich zu sprechen kommen werden, ist $\equiv_n$ mit

  $$\equiv_n = \{(a, b) \in \mathbb{Z}^2 \mid a \text{ und } b \text{ lassen bei Division durch } n \text{ denselben Rest}^1\}.$$

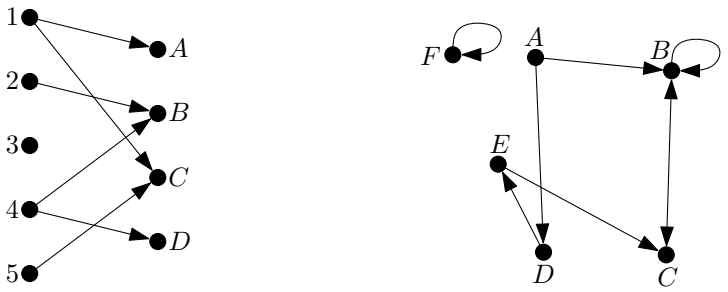
  Es gilt $5 \equiv_2 7$, $11 \equiv_3 2$, aber $8 \not\equiv_4 3$. Außerdem gilt $-1 \equiv_3 2$ und $-1 \equiv_2 1$.
- Für jede Menge M ist $\subseteq$ eine Relation auf $\mathcal{P}(M)$.
- Ein Beispiel für eine 3-stellige Relation zwischen Punkten in der Ebene ist

  $$\{(x, y, z) \in (\mathbb{R}^2)^3 \mid x, y \text{ und } z \text{ liegen auf einer Geraden}\}.$$

- Ein Beispiel für eine n-stellige Relation zwischen reellen Zahlen ist

  $$\{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_1 + \ldots + x_n = 0\}.$$

Im Folgenden werden wir uns hauptsächlich mit binären Relationen beschäftigen. Diese kann man gut durch Diagramme darstellen, in denen man die Elemente der Mengen durch Punkte darstellt und die Relationen durch Pfeile.



Die linke Abbildung stellt die Relation R zwischen den beiden Mengen $\{1, 2, 3, 4, 5\}$ und $\{A, B, C, D\}$ dar, wobei gilt

$$R = \{(1, A), (1, C), (2, B), (4, B), (4, D), (5, C)\}.$$

Die rechte Abbildung stellt die Relation Q auf der Menge $\{A, B, C, D, E, F\}$ dar, wobei gilt

$$Q = \{(A, B), (A, D), (B, B), (B, C), (C, B), (D, E), (E, C), (F, F)\}.$$

Man beachte, dass im rechten Diagramm der Pfeil zwischen B und C zwei Spitzen besitzt. Dies bedeutet, dass sowohl B zu C in Relation steht als auch C zu B.

Uns werden oft Relationen begegnen, die gewisse strukturelle Eigenschaften aufweisen. Die wichtigsten dieser Eigenschaften halten wir in der folgenden Definition fest.

Definition 2.11. *Es sei R eine binäre Relation auf einer Menge M.*

a) R ist reflexiv $\iff \forall a \in M : a\,R\,a$.

b) R ist symmetrisch $\iff \forall a, b \in M : (a\,R\,b \Rightarrow b\,R\,a)$.

[1] Eine Zahl $a \in \mathbb{Z}$ lässt den Rest $r \in \{0, 1, \ldots, n-1\}$ bei Division durch n, wenn es ein $x \in \mathbb{Z}$ gibt, für das $a = nx + r$ gilt.

c) R *ist* antisymmetrisch $\iff \forall a, b \in M : ((a\,R\,b \wedge b\,R\,a) \Rightarrow a = b)$.

d) R *ist* transitiv $\iff \forall a, b, c \in M : ((a\,R\,b \wedge b\,R\,c) \Rightarrow a\,R\,c)$.

e) R *ist eine* Äquivalenzrelation $\iff$ R *ist reflexiv, symmetrisch und transitiv.*

Man kann sich leicht überlegen, was diese Eigenschaften für das Diagramm der Relation R bedeuten. Ist R reflexiv, so besitzt jeder Punkt eine Schlinge, also einen Pfeil, der auf den Punkt zeigt, von dem er ausgeht. Ist R symmetrisch, so gibt es ausschließlich Doppelpfeile mit zwei Spitzen und Schlingen. Ist R antisymmetrisch, so gibt es keine Doppelpfeile. Eine Relation kann also gleichzeitig symmetrisch und antisymmetrisch sein. In diesem Fall gibt es nur Schlingen. Ist R transitiv, so besagt die obige Definition für alle $a, c \in M$ Folgendes: Gibt es ein $b \in M$, sodass es einen Pfeil von a zu b und einen Pfeil von b zu c gibt, so gibt es auch einen Pfeil von a zu c. Auf Äquivalenzrelationen und ihre Eigenschaften kommen wir in Abschnitt 2.4.3 noch ausführlich zu sprechen.

Beispiele

Der Leser sollte sich als Übung überlegen, warum die folgenden Aussagen zutreffen.

- Die Relation $<$ auf $\mathbb{R}$ ist nicht reflexiv, nicht symmetrisch, antisymmetrisch und transitiv. Man beachte, dass die Antisymmetrie trivialerweise folgt, da es gar keine Elemente $a, b \in \mathbb{R}$ mit $a < b$ und $b < a$ gibt. Die Relation $\leq$ auf $\mathbb{R}$ ist reflexiv, nicht symmetrisch, antisymmetrisch und transitiv.
- Es sei $n \in \mathbb{N}$ beliebig. Die Relation $\equiv_n$ auf $\mathbb{Z}$ ist eine Äquivalenzrelation (reflexiv, symmetrisch und transitiv), aber nicht antisymmetrisch.
- Es sei M eine beliebige Menge. Die Relation $\subseteq$ auf $\mathcal{P}(M)$ ist reflexiv, nicht symmetrisch, antisymmetrisch und transitiv.

2.4.2 Abbildungen

Betrachten wir noch einmal die Relation R mit

$$R = \{(s, u) \in S \times U \mid \text{Student } s \text{ besucht Übungsgruppe } u\}.$$

Diese besitzt (zumindest theoretisch) die Eigenschaft, dass jeder Student aus S zu genau einer Übungsgruppe aus U in Relation steht. Relationen mit dieser Eigenschaft heißen *Abbildungen* oder *Funktionen.*

Definition 2.12. *Eine Relation $f \subseteq A \times B$ heißt* Abbildung *oder* Funktion*, wenn jedes $a \in A$ zu genau einem Element $b \in B$ in Relation steht. Um anzudeuten, dass f eine Abbildung ist, schreiben wir $f\colon A \to B$ anstatt $f \subseteq A \times B$.*

1. *Für jedes $a \in A$ bezeichne $f(a)$ das eindeutige Element aus B, zu dem a in Relation steht. Wir sagen, dass a auf $f(a)$* abgebildet *wird.*

2. *Die Menge A heißt* Definitionsmenge *von f und die Menge B heißt* Zielmenge *von f. Die Menge*

$$f(A) = \{b \in B \mid \exists a \in A : f(a) = b\}$$

heißt Bildmenge *von f.*

3. *Die Menge $G(f) = \{(a, f(a)) \mid a \in A\} \subseteq A \times B$ heißt der* Graph *von f.*

4. *Für $A' \subseteq A$ definieren wir*

$$f(A') = \{b \in B \mid \exists a \in A' : f(a) = b\} \subseteq B$$

und für $B' \subseteq B$ definieren wir

$$f^{-1}(B') = \{a \in A \mid f(a) \in B'\} \subseteq A.$$

Abbildungen der Form $f\colon \mathbb{R} \to \mathbb{R}$ sind uns aus der Schule wohlbekannt. Um eine solche Abbildung zu beschreiben, gibt es zwei gängige Möglichkeiten. Möchte man beispielsweise die Abbildung beschreiben, die jedes $x \in \mathbb{R}$ auf $x^2 + 5$ abbildet, so schreibt man entweder $f(x) = x^2 + 5$ oder $f\colon x \mapsto x^2 + 5$ (nicht aber $f\colon x \to x^2 + 5$).

Beispiele

Beispiele für Abbildungen gibt es zur Genüge.

- $f\colon \mathbb{R} \to \mathbb{R}$ mit $f\colon x \mapsto x^3$
- floor$\colon \mathbb{R} \to \mathbb{R}$ mit floor$\colon x \mapsto$ größte ganze Zahl kleiner oder gleich x
  Es gilt z. B. floor(3,2) $= 3$, floor$(5) = 5$ und floor$(-1{,}2) = -2$.
- sgn$\colon \mathbb{Z} \to \{0, 1\}$ mit

$$\operatorname{sgn}(x) = \begin{cases} 0 & \text{falls } x \text{ gerade} \\ 1 & \text{falls } x \text{ ungerade} \end{cases}$$

- $g\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ mit $g\colon (x, y) \mapsto x + y$
- $h\colon \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ mit $h\colon x \mapsto (x, -x)$
- Sei M eine beliebige endliche Menge. Dann ist $f_1\colon \mathcal{P}(M) \to \mathbb{N}_0$ mit $f_1\colon N \mapsto |N|$ eine Abbildung, die jede Teilmenge von M auf ihre Kardinalität abbildet.
- Sei M eine beliebige Menge. Dann ist $f_2\colon \mathcal{P}(M) \to \mathcal{P}(M)$ mit $f_2\colon N \mapsto M \setminus N$ eine Abbildung, die jede Teilmenge von M auf ihr Komplement abbildet.

Bei Abbildungen muss man darauf achten, dass jedes Element auf genau ein anderes Element abgebildet wird. Dies ist zum Beispiel bei

$$f\colon \mathbb{N} \to \mathbb{R} \quad \text{mit} \quad f\colon x \mapsto y \in \mathbb{R} \text{ mit } y^2 = x$$

nicht der Fall, da 4 sowohl auf 2 als auch auf -2 abgebildet werden kann. Wir sagen dann, dass f *nicht wohldefiniert* ist. Ebenfalls nicht wohldefiniert ist

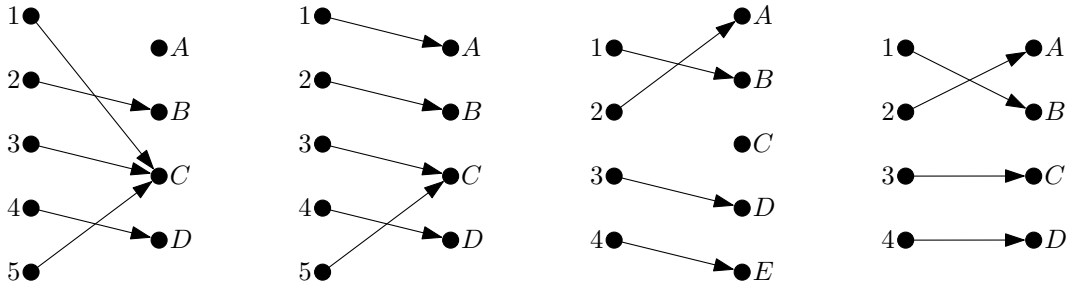
$$g\colon \mathbb{R} \to \mathbb{N}_0 \quad \text{mit} \quad g\colon x \mapsto \text{das kleinste } n \in \mathbb{N}_0 \text{ mit } \sum_{i=0}^{n} \frac{1}{2^i} > x,$$

da es z. B. für $x = 3$ gar kein n mit der entsprechenden Eigenschaft gibt (es gilt $\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$).

Definition 2.13. *Es sei $f\colon A \to B$ eine beliebige Abbildung.*

1. *f ist* injektiv $\iff \forall a, a' \in A : (f(a) = f(a') \Rightarrow a = a')$.
2. *f ist* surjektiv $\iff \forall b \in B : \exists a \in A : f(a) = b$.
3. *f ist* bijektiv $\iff$ *f ist injektiv und surjektiv.*

Eine injektive Abbildung $f\colon A \to B$ bildet verschiedene Elemente aus A auf verschiedene Elemente aus B ab. Bei einer surjektiven Abbildung gibt es für jedes Element $b \in B$ mindestens ein Element aus A, das auf b abgebildet wird. Bei einer bijektiven Abbildung treffen diese beiden Eigenschaften zusammen. Das bedeutet, dass für jedes Element $b \in B$ genau ein Element aus A existiert, das auf b abgebildet wird. Wir illustrieren diese Begriffe mit den folgenden Abbildungen.



nicht injektiv, nicht surjektiv | nicht injektiv, surjektiv | injektiv, nicht surjektiv | injektiv, surjektiv

Beispiele

Wir betrachten noch einmal die Abbildungen, die wir oben als Beispiele gegeben haben. Wieder überlassen wir es dem Leser, die folgenden Behauptungen nachzuprüfen.

- Die Funktion $f\colon \mathbb{R} \to \mathbb{R}$ mit $f\colon x \mapsto x^3$ ist bijektiv.
- Die Funktion floor ist nicht injektiv und nicht surjektiv. Sie wäre surjektiv, wenn wir sie als Funktion $f\colon \mathbb{R} \to \mathbb{Z}$ definiert hätten.
- Die Funktion sgn ist nicht injektiv, aber surjektiv.
- Die Funktion g ist nicht injektiv, aber surjektiv.
- Die Funktion h ist injektiv, aber nicht surjektiv.
- Die Funktion f_1 ist nicht injektiv und nicht surjektiv.
- Die Funktion f_2 ist bijektiv.

Wie im zweiten Beispiel bereits angedeutet, können wir eine Abbildung immer dadurch surjektiv machen, dass wir die Zielmenge auf die Bildmenge einschränken. Ist also $f\colon A \to B$ eine Funktion, so ist die Funktion $g\colon A \to f(A)$ mit $g\colon a \mapsto f(a)$ stets surjektiv.

Mithilfe von bijektiven Abbildungen können wir formal definieren, wann zwei Mengen dieselbe Kardinalität besitzen.

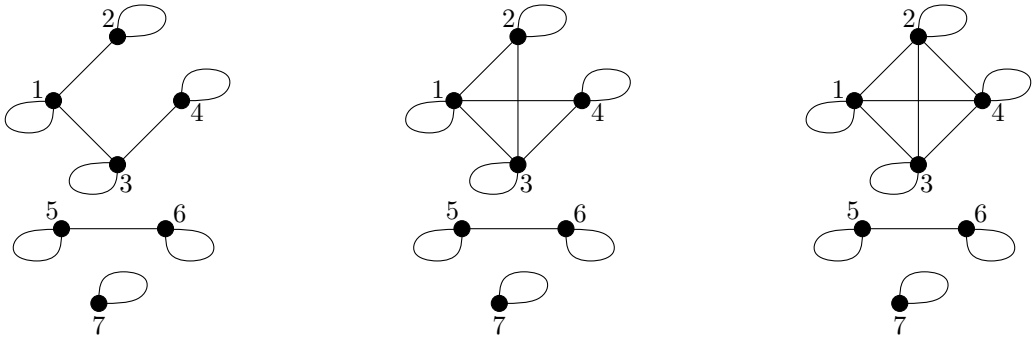
Definition 2.14. *Es seien A und B zwei beliebige Mengen. Wir sagen, dass A und B dieselbe Kardinalität besitzen, wenn es eine bijektive Abbildung $f\colon A \to B$ gibt. Wir nennen die Mengen A und B dann auch* gleichmächtig.

Für endliche Mengen entspricht diese Definition der natürlichen Sichtweise, dass zwei Mengen A und B dieselbe Kardinalität besitzen, wenn sie dieselbe Anzahl an Elementen enthalten. Der Leser sollte sich dies an einigen Beispielen klarmachen. Interessant ist die Definition aber für unendliche Mengen. Es stellt sich sofort die Frage, ob alle unendlichen Mengen dieselbe Kardinalität besitzen oder nicht. Mit dieser Frage werden wir uns in Abschnitt 4.1 noch ausführlich beschäftigen.

2.4.3 Äquivalenzrelationen

Wir betrachten nun Äquivalenzrelationen und ihre Eigenschaften im Detail. Im Folgenden sei R eine Äquivalenzrelation auf einer beliebigen Menge M. Gemäß Definition 2.11 ist R reflexiv, symmetrisch und transitiv. Zunächst überlegen wir uns anschaulich, wie das Diagramm von R aussieht. Reflexivität und Symmetrie bedeuten, dass es von jedem Element eine Schlinge zu sich selbst gibt und dass es abgesehen von diesen Schlingen nur Doppelpfeile mit zwei Spitzen gibt. Die Transitivität von R besagt, dass die Existenz von Pfeilen zwischen a und b sowie zwischen b und c auch die Existenz eines Pfeiles zwischen a und c impliziert.

Wir überlegen uns zunächst, welche Konsequenzen diese Eigenschaften zusammen haben. In den nachfolgenden Diagrammen haben wir die Pfeilspitzen weggelassen, da bei Äquivalenzrelationen grundsätzlich nur Doppelpfeile vorhanden sind. Als erstes betrachten wir die linke Relation. Bei dieser handelt es sich nicht um eine Äquivalenzrelation, da keine Transitivität gegeben ist. Um die Relation transitiv zu machen, müssen wir den Pfeil $2 \leftrightarrow 3$ (wegen der Pfeile $2 \leftrightarrow 1$ und $1 \leftrightarrow 3$) sowie den Pfeil $1 \leftrightarrow 4$ (wegen der Pfeile $1 \leftrightarrow 3$ und $3 \leftrightarrow 4$) ergänzen. Dann erhalten wir die Relation in der Mitte. Wir stellen fest, dass diese Relation ebenfalls nicht transitiv ist. Der Grund ist, dass wir nun auch die neuen Pfeile berücksichtigen müssen. Fügen wir nun auch noch den Pfeil $2 \leftrightarrow 4$ (wegen der Pfeile $2 \leftrightarrow 1$ und $1 \leftrightarrow 4$) ein, so erhalten wir die Relation auf der rechten Seite. Der Leser sollte sich davon überzeugen, dass es sich dabei um eine Äquivalenzrelation handelt.



Die Äquivalenzrelation auf der rechten Seite besitzt eine einfach zu beschreibende Struktur. Sie teilt die Elemente der Grundmenge in drei Klassen $\{1,2,3,4\}$, $\{5,6\}$ und $\{7\}$ ein. Dabei steht jedes Paar von Elementen aus derselben Klasse zueinander in Relation, während kein Paar von Elementen aus verschiedenen Klassen zueinander in Relation steht. Der Leser sollte sich anschaulich davon überzeugen, dass jede Äquivalenzrelation die ihr zugrunde liegende Grundmenge auf diese Weise in Klassen einteilt. Wir werden diese wichtige Eigenschaft nun auch formal beweisen.

Definition 2.15. *Es sei R eine Äquivalenzrelation auf einer Menge M. Für $a \in M$ bezeichnen wir mit*

$$[\![a]\!]_R = \{b \in M \mid b\,R\,a\}$$

die Äquivalenzklasse von a bezüglich R. *Wenn die Relation R aus dem Kontext hervorgeht, so schreiben wir auch einfach $[\![a]\!]$ statt $[\![a]\!]_R$.*

Für die Äquivalenzrelation, die in der obigen Abbildung auf der rechten Seite dargestellt ist, gilt beispielsweise $[\![3]\!] = \{1,2,3,4\}$, $[\![5]\!] = \{5,6\}$ und $[\![7]\!] = \{7\}$. Oben haben wir die Relation $\equiv_n$ auf $\mathbb{Z}$ mit

$$\equiv_n = \{(a,b) \in \mathbb{Z}^2 \mid a \text{ und } b \text{ lassen bei Division durch } n \text{ denselben Rest}\}$$

eingeführt. Für die Relation $\equiv_2$ gilt $[\![1]\!] = \{\ldots,-5,-3,-1,1,3,5,\ldots\}$. Für die Relation $\equiv_3$ gilt $[\![5]\!] = \{\ldots,-7,-4,-1,2,5,8,\ldots\}$.

Theorem 2.16. *Es sei R eine Äquivalenzrelation auf einer Menge M. Dann gelten die folgenden beiden Aussagen.*

a) Für alle $a, b \in M$ gilt entweder $[\![a]\!] = [\![b]\!]$ oder $[\![a]\!] \cap [\![b]\!] = \emptyset$.

b) Es gilt $M = \bigcup_{a \in M} [\![a]\!]$.

Beweis. Wir beginnen mit dem Beweis von Aussage a). Seien $a \in M$ und $b \in M$ beliebige Elemente. Wir machen eine Fallunterscheidung und betrachten zunächst den Fall, dass $a\,R\,b$ gilt. In diesem Fall gilt $[\![a]\!] = [\![b]\!]$. Um dies zu zeigen, wählen wir ein beliebiges Element $c \in [\![a]\!]$. Gemäß Definition 2.15 gilt $c\,R\,a$. Gemeinsam mit $a\,R\,b$ folgt aus der Transitivität von R, dass auch $c\,R\,b$, also $c \in [\![b]\!]$, gilt. Da c ein beliebiges Element aus $[\![a]\!]$ ist, folgt aus dieser Überlegung $[\![a]\!] \subseteq [\![b]\!]$. Ganz analog kann man argumentieren, dass auch $[\![b]\!] \subseteq [\![a]\!]$ gilt. Zusammen bedeutet das $[\![a]\!] = [\![b]\!]$.

Es bleibt, den Fall zu betrachten, dass $a\,R\,b$ nicht gilt. In diesem Fall gilt $[\![a]\!] \cap [\![b]\!] = \emptyset$. Dies folgt mit einem einfachen Widerspruchsbeweis. Angenommen es gibt ein Element $c \in [\![a]\!] \cap [\![b]\!]$. Für dieses Element gilt $c\,R\,a$ und $c\,R\,b$. Wegen der Symmetrie von R gilt auch $b\,R\,c$ und damit folgt insgesamt wegen der Transitivität von R, dass $b\,R\,a$ und damit auch $a\,R\,b$ gilt. Dies ist ein Widerspruch zu der Annahme, dass a und b nicht zueinander in Relation stehen. Der Beweis von a) ist damit abgeschlossen

Um b) zu beweisen, zeigen wir zunächst, dass $M \subseteq \bigcup_{a \in M} [\![a]\!]$ gilt. Sei $b \in M$ beliebig. Wegen der Reflexivität von R gilt $b\,R\,b$ und damit auch $b \in [\![b]\!]$. Also gilt $b \in \bigcup_{a \in M} [\![a]\!]$. Andersherum gilt per Definition $[\![a]\!] \subseteq M$ für jedes $a \in M$. Also gilt auch $\bigcup_{a \in M} [\![a]\!] \subseteq M$. Damit ist Teil b) bewiesen. □

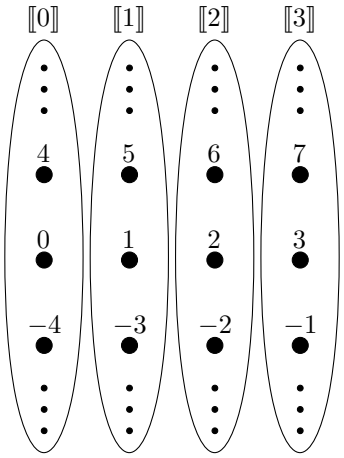
Aus dem vorangegangenen Theorem folgt direkt die oben bereits diskutierte Eigenschaft, dass eine Äquivalenzrelation R ihre Grundmenge M in disjunkte Klassen einteilt (eine solche Einteilung einer Menge nennt man auch eine *Partition* der Menge). Dabei kann jedes Element $a \in M$ als *Repräsentant* seiner Klasse $[\![a]\!]$ angesehen werden. Bei der Relation $\equiv_3$ repräsentiert beispielsweise die Zahl 1 die Klasse der Zahlen, die bei Division durch 3 den Rest 1 lassen. Dieselbe Klasse wird zum Beispiel auch von den Zahlen 4 und 7 repräsentiert.

Wir betrachten als weiteres Beispiel eine Relation auf S, der Menge aller Teilnehmer dieser Vorlesung. Wir sagen, dass zwei Personen $s \in S$ und $t \in S$ zueinander in Relation R stehen, wenn sie am gleichen Tag des Jahres Geburtstag haben. Man überzeugt sich leicht davon, dass R eine Äquivalenzrelation ist. Durch diese Relation wird die Menge S in höchstens 366 Klassen eingeteilt. Die tatsächliche Zahl der Klassen ist wahrscheinlich kleiner als die Anzahl der Tage im Jahr, da es Tage gibt, an denen keiner der Teilnehmer Geburtstag hat.

Wir betrachten nun noch einmal die wichtige Relation $\equiv_n$ auf $\mathbb{Z}$. Wir bezeichnen mit $\mathbb{Z}/n\mathbb{Z}$ die Menge der Äquivalenzklassen dieser Relation. Es gilt

$$\mathbb{Z}/n\mathbb{Z} = \{[\![0]\!], [\![1]\!], [\![2]\!], \ldots, [\![n-1]\!]\}.$$

Der Leser sollte sich klar machen, dass die Objekte in der Menge $\mathbb{Z}/n\mathbb{Z}$ keine Zahlen, sondern Äquivalenzklassen sind. Die folgende Abbildung illustriert noch einmal die Menge $\mathbb{Z}/4\mathbb{Z} = \{[\![0]\!], [\![1]\!], [\![2]\!], [\![3]\!]\}$.



Zunächst handelt es sich bei $\mathbb{Z}/n\mathbb{Z}$ lediglich um eine Menge ohne weitere Struktur. Wir definieren zwei Verknüpfungen auf dieser Menge, die die Grundlage der sogenannten *modularen Arithmetik* bilden.

Definition 2.17. *Es sei $n \in \mathbb{N}$ beliebig. Wir definieren die* Addition modulo n *durch eine Funktion $\oplus_n \colon \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$. Für zwei Äquivalenzklassen $A, B \in \mathbb{Z}/n\mathbb{Z}$ schreiben wir $A \oplus_n B$ statt $\oplus_n(A, B)$ und wir definieren $[\![a]\!] \oplus_n [\![b]\!] = [\![a + b]\!]$ für alle $a, b \in \{0, 1, \ldots, n-1\}$.*

Wir definieren die Multiplikation modulo n *durch eine Funktion $\odot_n \colon \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$. Für zwei Äquivalenzklassen $A, B \in \mathbb{Z}/n\mathbb{Z}$ schreiben wir $A \odot_n B$ statt $\odot_n(A, B)$ und wir definieren $[\![a]\!] \odot_n [\![b]\!] = [\![a \cdot b]\!]$ für alle $a, b \in \{0, 1, \ldots, n-1\}$.*

Hätten wir in der vorangegangenen Definition statt $a, b \in \{0, 1, \ldots, n-1\}$ auch $a, b \in \mathbb{Z}$ schreiben dürfen? Diese Frage können wir nur mit jein beantworten. Hätten wir dies getan, hätte die Gefahr bestanden, dass die Funktionen $\oplus_n$ und $\odot_n$ nicht wohldefiniert sind. Dies liegt daran, dass es verschiedene Elemente aus $\mathbb{Z}$ gibt, die dieselbe Äquivalenzklasse repräsentieren. Das folgende Theorem besagt jedoch, dass die Verknüpfungen $\oplus_n$ und $\odot_n$ auch für $a, b \in \mathbb{Z}$ wohldefiniert sind.

Theorem 2.18. *Es sei $n \in \mathbb{N}$ beliebig und es seien $a, a', b, b' \in \mathbb{Z}$ so gewählt, dass $[\![a]\!] = [\![a']\!]$ und $[\![b]\!] = [\![b']\!]$ gilt. Dann gilt*

$$[\![a + b]\!] = [\![a' + b']\!] \quad \textit{und} \quad [\![a \cdot b]\!] = [\![a' \cdot b']\!].$$

Beweis. Aufgrund von $[\![a]\!] = [\![a']\!]$ und $[\![b]\!] = [\![b']\!]$ gibt es $r, q \in \{0, 1, 2, \ldots, n-1\}$ und $x, x', y, y' \in \mathbb{Z}$ mit $a = xn + r$, $a' = x'n + r$, $b = yn + q$ und $b' = y'n + q$. Demzufolge gilt

$$a + b = (xn + r) + (yn + q) = (x + y)n + (r + q)$$

und

$$a' + b' = (x'n + r) + (y'n + q) = (x' + y')n + (r + q).$$

Also lassen sowohl $a + b$ als auch $a' + b'$ denselben Rest bei Division durch n. Daraus folgt $a + b \equiv_n a' + b'$, was gemäß Theorem 2.16 $[\![a + b]\!] = [\![a' + b']\!]$ bedeutet.

Außerdem gilt

$$a \cdot b = (xn + r) \cdot (yn + q) = (xyn + xq + yr)n + (rq)$$

und

$$a' \cdot b' = (x'n + r) \cdot (y'n + q) = (x'y'n + x'q + y'r)n + (rq).$$

Also lassen sowohl $a \cdot b$ als auch $a' \cdot b'$ denselben Rest bei Division durch n. Daraus folgt $a \cdot b \equiv_n a' \cdot b'$, was gemäß Theorem 2.16 $[\![a \cdot b]\!] = [\![a' \cdot b']\!]$ bedeutet. □

Modulare Arithmetik begegnet uns auch im Alltag. Die Tageszeit ist beispielsweise eine Addition modulo 24, denn 10 Stunden nach 20 Uhr ist es 6 Uhr, wie die folgende Rechnung zeigt $[\![20]\!] \oplus_{24} [\![10]\!] = [\![6]\!]$. Ebenso ist modulare Arithmetik bei Prozessoren und beim Programmieren wichtig. Haben wir eine Variable oder ein Register mit 8 Bits, so können wir damit genau die natürlichen Zahlen zwischen 0 und 255 darstellen. Die herkömmlichen implementierten Additions- und Multiplikationsoperationen, die Überläufe nicht berücksichtigen, entsprechen modularer Arithmetik. Beispielsweise ergibt die Summe von 10 und 246 wegen des Überlaufes 0. Dies entspricht der Rechnung $[\![10]\!] \oplus_{256} [\![246]\!] = [\![0]\!]$.

Kapitel 3

Endliche Automaten und formale Sprachen

In der theoretischen Informatik arbeitet man mit *abstrakten Rechnermodellen*. Analysiert man beispielsweise die Laufzeit eines Algorithmus, so bezieht man sich dabei nicht auf irgendeinen konkreten Rechner, da die verfügbare Hardware sich so schnell ändert, dass solche Analysen schnell an Bedeutung verlieren würden. Stattdessen betrachtet man mathematische Modelle, die nur die wesentlichen Aspekte realer Rechner abbilden und auf diese Weise von der konkreten Hardware abstrahieren. Dies hat den Vorteil, dass Analysen, die auf solchen Modellen basieren, dauerhaft Bestand haben.

In diesem Kapitel betrachten wir das abstrakte Rechnermodell der *endlichen Automaten*. Dieses Modell eignet sich gut zum Einstieg, da es relativ einfach ist. Es wurde nicht mit dem Ziel entworfen, die Mächtigkeit realer Rechner abzubilden, sondern für andere Zwecke. Ein endlicher Automat erhält als Eingabe eine Zeichenkette und gibt nach deren Verarbeitung entweder *Ja* oder *Nein* aus. Einen solchen Automaten kann man beispielsweise nutzen, um zu entscheiden, ob eine gegebene Zeichenkette ein gültiger Variablenname in Java ist.

Eng verbunden mit der Theorie von endlichen Automaten sind *formale Sprachen*. Eine solche Sprache ist nichts anderes als eine Menge von Zeichenketten über einem gegebenen Alphabet. So bildet zum Beispiel die Menge aller gültigen Java-Klassen eine formale Sprache ebenso wie die Menge aller gültigen HTML-Dateien. Für den Entwurf von Programmiersprachen und Compilern ist es unerlässlich eine formale Beschreibung zu entwickeln, welche Zeichenfolgen ein gültiges Programm darstellen. Diese Beschreibung muss dergestalt sein, dass der Compiler möglichst effizient die *lexikalische Analyse* und die *syntaktische Analyse* durchführen kann. Die lexikalische Analyse ist der erste Schritt, den ein Compiler durchführt; dabei wird der Quelltext in logisch zusammenhängende *Tokens* wie zum Beispiel Schlüsselwörter, Zahlen und Operatoren zerlegt. Der zweite Schritt ist die syntaktische Analyse, in der überprüft wird, ob der Quelltext ein syntaktisch korrektes Programm ist, und in der der Quelltext in einen sogenannten *Syntaxbaum* umgewandelt wird.

Wir werden in diesem Kapitel zunächst *Grammatiken* kennenlernen. Dabei handelt es sich um Regelsysteme, die beschreiben, wie die Wörter einer Sprache erzeugt werden. Grammatiken werden gemäß ihrer Mächtigkeit in verschiedene Klassen eingeteilt. Wir

werden uns in dieser Vorlesung mit *regulären Grammatiken* im Detail beschäftigen und in der Vorlesung „Algorithmen und Berechnungskomplexität I“ auf *kontextfreie Grammatiken* eingehen. Während reguläre Grammatiken bei der lexikalischen Analyse eine große Rolle spielen, kommen kontextfreie Grammatiken bei der Syntaxanalyse zum Einsatz, da alle gängigen Programmiersprachen durch kontextfreie Grammatiken beschrieben werden können, wenn man von einigen nicht ganz so wichtigen Details absieht.

3.1 Sprachen und Grammatiken

In diesem Kapitel gehen wir stets davon aus, dass Σ eine beliebige endliche Menge ist, die wir auch das *Alphabet* nennen werden. Ein *Wort* ist eine endliche Folge von Zeichen aus dem Alphabet Σ, und mit Σ^* bezeichnen wir die Menge aller Wörter, das heißt die Menge aller endlichen Zeichenfolgen über Σ. Das *leere Wort* ε ist die Zeichenkette der Länge 0, und wir definieren, dass ε zu Σ^* gehört. Wir definieren außerdem $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$ als die Menge aller Wörter mit mindestens einem Buchstaben. Nun können wir das zentrale Konzept dieses Kapitels definieren.

Definition 3.1. *Eine Menge $L \subseteq \Sigma^*$ bezeichnen wir als* (formale) Sprache über Σ.

Wir benötigen noch die folgenden Schreibweisen.

Definition 3.2. *Sei Σ ein Alphabet, $a \in \Sigma$ und $n \in \mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$. Mit a^n bezeichnen wir das Wort $a \cdots a$, in dem der Buchstabe a genau n mal wiederholt wird. Außerdem bezeichnet a^0 das leere Wort ε. Sei $w \in \Sigma^*$ ein Wort, dann bezeichnet $|w|$ die Länge von w und $|w|_a$ gibt an, wie oft der Buchstabe a in dem Wort w enthalten ist. Außerdem bezeichnet w^R das gespiegelte Wort, das heißt für $w = w_1 \ldots w_n$ ist $w^R = w_n \ldots w_1$.*

Fangen wir mit dem einfachen Beispiel $\Sigma = \{1\}$ an, bei dem das Alphabet aus nur einem einzigen Element besteht. Für dieses Beispiel gilt $\Sigma^* = \{\varepsilon, 1, 11, 111, 1111, \ldots\} = \{1^0, 1^1, 1^2, 1^3, \ldots\}$. Sprachen über diesem Alphabet sind zum Beispiel $L = \{1^n \mid n \text{ ist gerade}\}$ und $L = \{1^n \mid n \text{ ist eine Primzahl}\}$.

Ein bei Informatikern sehr beliebtes Alphabet ist $\Sigma = \{0, 1\}$. Über diesem Alphabet kann man beispielsweise die folgenden Sprachen definieren.

- $L = \{01, 1111, 010, \varepsilon\}$
  Eine Sprache, die vier willkürlich gewählte Wörter enthält.
- $L = \{0^m 1^n \mid m, n \in \mathbb{N}_0\}$
  Die Sprache aller Wörter, die aus einem Block von Nullen und einem Block von Einsen bestehen, die aufeinanderfolgen.
- $L = \{0^n 1^n \mid n \in \mathbb{N}_0\}$
  Die Sprache aller Wörter, die aus einem Block von Nullen und einem Block von Einsen bestehen, die aufeinanderfolgen und die gleiche Länge haben.

- $L = \{w \in \Sigma^* \mid |w|_0 = |w|_1\}$
  Die Sprache aller Wörter, die genauso viele Nullen wie Einsen enthalten.
- $L = \{w \in \Sigma^* \mid w = w^R\}$
  Die Sprache aller *Palindrome*.

Eine geeignete Wahl für Programmiersprachen ist es, als Alphabet Σ die Menge aller Unicode-Zeichen zu wählen. Dann kann man L zum Beispiel als die Menge aller syntaktisch korrekten Java-Klassen definieren.

Um Sprachen kompakt zu beschreiben, werden wir Grammatiken verwenden.

Definition 3.3. *Eine* Grammatik G *besteht aus vier Komponenten* (Σ, V, S, P) *mit den folgenden Bedeutungen:*

- Σ *ist das endliche Alphabet, über dem die Sprache definiert ist. Wir nennen die Elemente aus* Σ *auch* Terminalsymbole.
- V *ist eine endliche Menge von* Nichtterminalsymbolen*, die disjunkt zu* Σ *ist.*
- $S \in V$ *ist das* Startsymbol.
- P *ist eine endliche Menge von* Ableitungsregeln*. Dabei ist eine Ableitungsregel ein Paar* (ℓ, r) *mit* $\ell \in V^+$ *und* $r \in (V \cup \Sigma)^*$*. Wir werden statt* (ℓ, r) *auch* $\ell \to r$ *schreiben. Enthält* P *die Regeln* $(\ell, r_1), \ldots, (\ell, r_k)$*, so schreiben wir auch* $\ell \to r_1, \ldots, r_k$*.*

Wörter werden nun wie folgt erzeugt. Man startet mit dem Startsymbol S und wendet solange Ableitungsregeln an, bis nur noch Terminale vorhanden sind. Das Anwenden einer Ableitungsregel (ℓ, r) bedeutet, dass wir ein Vorkommen von ℓ durch r ersetzen.

Definition 3.4. *Für eine Grammatik* $G = (\Sigma, V, S, P)$ *bezeichnen wir mit* $L(G) \subseteq \Sigma^*$ *die von ihr erzeugte Sprache, das heißt die Menge aller Wörter, die aus dem Startsymbol mithilfe der Ableitungsregeln aus* P *erzeugt werden können.*

Schauen wir uns Beispiele für Grammatiken an, um diese Definition zu verdeutlichen. Dabei sei stets $\Sigma = \{0, 1\}$ angenommen. Formale Beweise, dass die Grammatiken die behaupteten Sprachen erzeugen, überlassen wir dem Leser.

- Es sei $V = \{S\}$ und P enthalte die Regeln $S \to \varepsilon$ und $S \to 11S$. Für diese Grammatik G gilt $L(G) = \{1^n \mid n \in \mathbb{N}_0 \text{ und } n \text{ ist gerade}\}$. Eine Ableitung des Wortes 1111 sieht zum Beispiel wie folgt aus:

  $$S \to 11S \to 1111S \to 1111\varepsilon = 1111.$$

- Es sei $V = \{S, A, B\}$ und P enthalte die Regeln

  $$S \to AB \qquad A \to \varepsilon, 0A \qquad B \to \varepsilon, 1B.$$

  Für diese Grammatik G gilt $L(G) = \{0^m 1^n \mid m, n \in \mathbb{N}\}$. Eine Ableitung des Wortes 011 sieht zum Beispiel wie folgt aus:

  $$S \to AB \to 0AB \to 0\varepsilon B = 0B \to 01B \to 011B \to 011\varepsilon = 011.$$

- Es sei $V = \{S\}$ und P enthalte die Regeln

$$S \to \varepsilon, 0S1.$$

Für diese Grammatik G gilt $L(G) = \{0^n 1^n \mid n \in \mathbb{N}\}$. Eine Ableitung des Wortes 0011 sieht zum Beispiel wie folgt aus:

$$S \to 0S1 \to 00S11 \to 00\varepsilon 11 = 0011.$$

- Es sei $V = \{S\}$ und P enthalte die Regeln

$$S \to \varepsilon, 0, 1, 0S0, 1S1.$$

Für diese Grammatik G gilt $L(G) = \{w \in \Sigma^* \mid w = w^R\}$. Wir geben beispielhaft eine Ableitung des Palindroms 1101011 an:

$$S \to 1S1 \to 11S11 \to 110S011 \to 1101011.$$

Für eine gegebene Grammatik und ein gegebenes Wort ist es ein wichtiges Problem, zu entscheiden, ob das Wort zu der von der Grammatik erzeugten Sprache gehört, ob also zum Beispiel ein gegebener Quelltext ein syntaktisch korrektes Programm einer bestimmten Programmiersprache darstellt. Dieses Problem wird auch *Wortproblem* genannt und es kann für allgemeine Grammatiken so komplex sein, dass es keine Algorithmen gibt, um es zu lösen. Deshalb müssen wir die erlaubten Ableitungsregeln einschränken, um eine Klasse von Grammatiken zu erhalten, für die das Wortproblem effizient gelöst werden kann. Auf der anderen Seite müssen wir natürlich aufpassen, dass wir die Grammatiken nicht zu stark einschränken, weil wir ansonsten keine mächtigen Programmiersprachen wie C++ oder Java beschreiben können. Die Chomsky-Hierarchie enthält vier Klassen, die verschiedene Kompromisse zwischen der Ausdrucksstärke der Grammatik und der Komplexität des Wortproblems darstellen.

Definition 3.5. *Die* Chomsky-Hierarchie *teilt Grammatiken in die folgenden vier Klassen ein.*

0. *Grammatiken ohne Einschränkungen heißen* Chomsky-0-Grammatiken.

1. *In* Chomsky-1-Grammatiken *oder* kontextsensitiven Grammatiken *haben alle Ableitungsregeln die Form $\alpha A\beta \to \alpha\gamma\beta$ oder $S \to \varepsilon$, wobei $\alpha, \beta \in V^*$, $A \in V$ und $\gamma \in (V \cup \Sigma)^+$ gilt. Außerdem darf S auf keiner rechten Seite einer Ableitungsregel vorkommen.*

2. *In* Chomsky-2-Grammatiken *oder* kontextfreien Grammatiken *haben alle Ableitungsregeln die Form $A \to v$ mit $A \in V$ und $v \in (V \cup \Sigma)^*$.*

3. *In* Chomsky-3-Grammatiken *oder* rechtslinearen Grammatiken *oder* regulären Grammatiken *haben alle Ableitungsregeln die Form $A \to v$ mit $A \in V$ und $v = \varepsilon$ oder $v = aB$ mit $a \in \Sigma$ und $B \in V$.*

Die Chomsky-Hierarchie wurde nach dem berühmten Linguisten Noam Chomsky benannt, der sie 1956 eingeführt hat.

In dieser Vorlesung sind nur die regulären Grammatiken von Interesse. Die kontextfreien Grammatiken werden wir in der Vorlesung „Algorithmen und Berechnungskomplexität I“ im Detail besprechen und die anderen beiden Klassen haben wir nur der Vollständigkeit halber aufgeführt. Wir werden Sprachen, die von regulären oder kontextfreien Grammatiken erzeugt werden, im Folgenden auch *reguläre Sprachen* bzw. *kontextfreie Sprachen* nennen.

3.2 Endliche Automaten

In diesem Abschnitt werden wir sehen, dass das Wortproblem für reguläre Grammatiken sehr effizient gelöst werden kann. Dazu führen wir mit *endlichen Automaten* ein einfaches Rechnermodell ein, und wir zeigen, dass die Sprachen, die von diesem einfachen Rechnermodell entschieden werden können, genau die regulären Sprachen sind. Bevor wir den Zusammenhang zu regulären Grammatiken herstellen, beschäftigen wir uns zunächst im Detail mit endlichen Automaten, die auch für sich genommen ein interessantes Rechnermodell darstellen, das beispielsweise als Modell sequentieller Schaltwerke dient.

Definition 3.6. *Ein* endlicher Automat *(deterministic finite automaton, DFA) M besteht aus fünf Komponenten $(Q, \Sigma, \delta, q_0, F)$ mit den folgenden Bedeutungen:*

- *Q ist eine endliche Menge, die* Zustandsmenge.
- *Σ ist eine endliche Menge, das* Eingabealphabet.
- *$\delta : Q \times \Sigma \to Q$ ist die* Zustandsüberführungsfunktion.
- *$q_0 \in Q$ ist der* Startzustand.
- *$F \subseteq Q$ ist die Menge der* akzeptierenden Zustände.

Ein solcher DFA M erhält als Eingabe ein Wort $w = w_1 \dots w_n$ über dem Alphabet Σ und arbeitet die Buchstaben von links nach rechts ab. Er startet im Startzustand q_0 und wechselt nach dem Lesen des ersten Buchstabens in den Zustand $q_1 = \delta(q_0, w_1)$. Nach dem Lesen des zweiten Buchstabens wechselt der DFA in den Zustand $q_2 = \delta(q_1, w_2)$ und so weiter. Nach dem Lesen des letzten Buchstabens stoppt der DFA im Zustand $q_n = \delta(q_{n-1}, w_n)$.

Da uns in der Regel nur der letzte erreichte Zustand q_n interessiert, definieren wir eine Funktion $\delta^* : Q \times \Sigma^* \to Q$, die jedes Paar (q, w) bestehend aus einem Zustand q und einem Wort $w \in \Sigma^*$ auf den Zustand abbildet, den der DFA erreicht, wenn er im Zustand q startet und das Wort w liest. Diese Funktion können wir als eine Erweiterung von δ auffassen und wir definieren

$$\delta^*(q, \varepsilon) = q \quad \text{und} \quad \delta^*(q, a) = \delta(q, a)$$

für alle $q \in Q$ und $a \in \Sigma$. Damit haben wir das Verhalten der Funktion δ^* auf allen Wörtern der Länge null und eins definiert. Das Verhalten für einen Zustand q und ein Wort $w = w_1 w_2$ der Länge zwei definieren wir durch

$$\delta^*(q, w_1 w_2) = \delta^*(\delta(q, w_1), w_2).$$

Bei dieser Definition nutzen wir aus, dass δ^* für das Wort w_2 der Länge eins bereits definiert ist. Allgemein definieren wir für jeden Zustand $q \in Q$ und für jedes Wort $w = w_1 \dots w_n \in \Sigma^*$ der Länge $n \geq 2$

$$\delta^*(q, w) = \delta^*(\delta(q, w_1), w_2 \dots w_n),$$

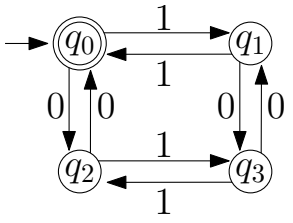
wobei wir davon ausgehen, dass δ^* bereits für das Wort $w_2 \dots w_n$ der Länge $n-1$ definiert ist. Dieses Vorgehen sollte den Leser an das Prinzip der vollständigen Induktion erinnern. Auch hier haben wir δ^* zunächst explizit für Wörter der Länge null und eins definiert (vergleichbar mit dem Induktionsanfang) und anschließend haben wir bei der Definition für Wörter der Länge $n \geq 2$ auf die Definition für Wörter der Länge $n-1$ zurückgegriffen (vergleichbar mit dem Induktionsschritt).

Erhält der DFA nun ein Wort $w \in \Sigma^*$ als Eingabe, so terminiert er im Zustand $q_n = \delta^*(q_0, w)$. Falls $q_n \in F$ gilt, sagen wir, dass der DFA das Wort w *akzeptiert*. Gilt $q_n \notin F$, so sagen wir, dass der DFA das Wort w *verwirft*. Mit

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

bezeichnen wir für einen DFA M die Menge aller Wörter, die M akzeptiert. Wir sagen, dass M die Sprache $L(M)$ *entscheidet* oder *akzeptiert*. Wir sagen auch, dass M ein DFA für die Sprache $L(M)$ ist.

Anschaulich kann man einen DFA anhand eines Übergangsgraphen darstellen. Das folgende Beispiel, in dem $\Sigma = \{0, 1\}$ gilt, ist aus dem Buch von Ingo Wegener übernommen [10].



Jeder Zustand aus Q ist durch einen Kreis dargestellt, Zustände aus F sind durch doppelte Kreise dargestellt und der Startzustand ist durch einen eingehenden Pfeil markiert, der von keinem anderen Zustand ausgeht. Die Funktion δ wird durch die Pfeile zwischen den Zuständen dargestellt. Liest der DFA beispielsweise im Zustand q_0 eine 1, so geht er in den Zustand q_1 über. Liest er im Zustand q_0 eine 0, so geht er in den Zustand q_2 über.

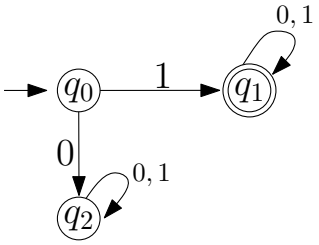
Nun sollten wir noch klären, welche Sprache der Automat M in diesem Beispiel entscheidet. Zunächst halten wir fest, dass q_0 der einzige akzeptierende Zustand ist. Das

heißt, nur wenn der DFA nach dem Lesen aller Zeichen des Eingabewortes w wieder im Startzustand ist, so akzeptiert er w. Man kann nun weiterhin beobachten, dass der Automat sich genau dann in einem der beiden linken Zustände (q_0 oder q_2) befindet, wenn er eine gerade Anzahl an Einsen gelesen hat. Weiterhin befindet sich der Automat genau dann in einem der beiden oberen Zustände (q_0 oder q_1), wenn er eine gerade Anzahl an Nullen gelesen hat. Der DFA befindet sich also genau dann im Zustand q_0, wenn er eine gerade Anzahl an Nullen und eine gerade Anzahl an Einsen gelesen hat. Es gilt also

$$L(M) = \{w \in \Sigma^* \mid |w|_0 \text{ ist gerade und } |w|_1 \text{ ist gerade}\}.$$

Einen formalen Beweis dieser Behauptung überlassen wir auch hier dem Leser.

Es kann vorkommen, dass ein DFA beim Lesen eines Zeichens gar keinen Zustandswechsel durchführt. In der grafischen Darstellung bedeutet das, dass es Schleifen gibt, die von einem Knoten zu sich selbst führen. Der folgende DFA akzeptiert beispielsweise genau diejenigen Wörter, die mit einer Eins beginnen.



Im Folgenden werden wir untersuchen, welche Sprachen von DFAs entschieden werden können, und wir werden mit *nichtdeterministischen Automaten* eine Erweiterung von DFAs kennenlernen.

3.2.1 Pumping-Lemma für endliche Automaten

Eine wichtige Frage für ein Rechnermodell ist, wie mächtig es ist, das heißt, welche Sprachen es entscheiden kann. In diesem Abschnitt möchten wir herausfinden, welche Sprachen von DFAs entschieden werden können und welche zu komplex für DFAs sind. Dazu werden wir das sogenannte *Pumping-Lemma* kennenlernen, mit dessen Hilfe wir zeigen können, dass es für manche Sprachen keinen DFA gibt.

Fangen wir mit der Sprache $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ an, die wir bereits am Anfang dieses Kapitels als Beispiel gesehen haben. Der Leser sollte an dieser Stelle selbst versuchen, einen DFA für L zu konstruieren. Er wird dabei auf folgendes Problem stoßen. Solange der DFA Nullen liest, muss er sich die Anzahl der bereits gelesenen Nullen merken, um sie später mit der Anzahl der gelesenen Einsen vergleichen zu können. Ein DFA hat aber per Definition nur eine endliche Zustandsmenge und er kann deshalb die Anzahl der Nullen für beliebig lange Wörter nicht korrekt zählen. Bei dieser einfachen Sprache können wir diese Intuition formalisieren und zeigen, dass es keinen DFA gibt, der L entscheidet.

Theorem 3.7. *Es gibt keinen DFA, der die Sprache* $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ *entscheidet.*

Beweis. Wir führen einen Widerspruchsbeweis und gehen davon aus, dass es einen solchen DFA M gibt. Es sei Q die endliche Zustandsmenge von M. Wir betrachten die Menge der Wörter $\{\varepsilon, 0, 0^2, \ldots, 0^{|Q|}\}$. Da es sich hierbei um mehr Wörter als Zustände handelt, muss es zwei Wörter 0^i und 0^j mit $i \neq j$ aus dieser Menge geben, nach deren Lesen M im gleichen Zustand ist, für die also $\delta^*(q_0, 0^i) = \delta^*(q_0, 0^j)$ gilt. Da M die Sprache L entscheidet, muss $\delta^*(q_0, 0^i 1^i) \in F$ gelten. Damit gilt auch

$$\delta^*(q_0, 0^j 1^i) = \delta^*(\delta^*(q_0, 0^j), 1^i) = \delta^*(\delta^*(q_0, 0^i), 1^i) = \delta^*(q_0, 0^i 1^i) \in F.$$

Damit akzeptiert M auch das Wort $0^j 1^i$. Wegen $j \neq i$ ist dies ein Widerspruch dazu, dass der DFA M die Sprache L entscheidet. □

Im Allgemeinen ist es schwierig oder zumindest aufwändig für eine Sprache auf diese oder ähnliche Weise nachzuweisen, dass sie von keinem DFA entschieden wird. Das folgende Pumping-Lemma liefert hingegen eine notwendige Bedingung dafür, dass L von einem DFA entschieden werden kann. Für viele Sprachen kann leicht gezeigt werden, dass sie diese Bedingung nicht erfüllen, woraus direkt folgt, dass sie von keinem DFA entschieden werden können. Mit dem Pumping-Lemma kann man also für viele Sprachen einfach zeigen, dass sie von keinem DFA entschieden werden. Es trägt seinen Namen, da es zeigt, dass man bei jedem Wort $w \in L$, das zu einer Sprache gehört, die von einem DFA entschieden wird, gewisse Teile vervielfältigen kann ohne dabei die Sprache L zu verlassen. Man kann also in gewisser Weise die Wörter aus L „aufpumpen“.

Lemma 3.8 (Pumping-Lemma)**.** *Sei L eine Sprache, die von einem DFA entschieden werden kann. Dann gibt es eine Konstante n, sodass für alle $z \in L$ mit $|z| \geq n$ gilt: Das Wort z kann in drei Teile $z = uvw$ mit $|uv| \leq n$ und $|v| \geq 1$ zerlegt werden, sodass $uv^i w \in L$ für alle $i \geq 0$ gilt.*

Beweis. Es sei M ein DFA für die Sprache L mit der Zustandsmenge Q und es sei $n = |Q|$ die Anzahl der Zustände von M. Beim Lesen eines Wortes $z \in L$ der Länge mindestens n durchläuft der DFA M eine Folge von mindestens $n+1$ Zuständen. Da der DFA nur n Zustände besitzt, muss mindestens einer von diesen mehrfach in dieser Folge auftreten. Wir bezeichnen mit q einen solchen Zustand und zwar denjenigen, dessen erste Wiederholung von allen mehrfach auftretenden Zuständen als erstes erfolgt.

Außerdem sei u das Präfix von z, nach dessen Lesen M zum ersten Mal in Zustand q ist, und es sei uv das Präfix von z, nach dessen Lesen M zum zweiten Mal in Zustand q ist. Dann gilt gemäß obiger Argumentation $|uv| \leq n$ und $|v| \geq 1$. Das Wort w sei nun einfach so gewählt, dass $z = uvw$ gilt. Die zentrale Eigenschaft dieser Wahl ist, dass M, wenn er im Zustand q startet und das Wort v liest, zum Zustand q zurückkehrt. Aus der Wahl von v folgt nämlich $\delta^*(q, v) = q$. Außerdem gilt $\delta^*(q_0, u) = q$.

Sei $q' = \delta^*(q_0, z) \in F$ der Zustand, den M nach dem Lesen von uvw erreicht. Da M sich nach dem Lesen des Präfix uv in Zustand q befindet, können wir diesen Zustand

auch als $q' = \delta^*(q, w)$ schreiben. Für die obige Wahl von u, v und w und für alle $i \geq 0$ gilt $uv^iw \in L$, denn

$$\begin{aligned} &\delta^*(q_0, uv^iw) = \delta^*(\delta^*(q_0, u), v^iw) \\ &= \delta^*(q, v^iw) = \delta^*(\delta^*(q, v), v^{i-1}w) = \delta^*(q, v^{i-1}w) = \ldots = \delta^*(q, vw) = \delta^*(q, w) \\ &= q' \in F. \end{aligned}$$

Stellen wir uns den DFA wieder als Graphen vor, so läuft dieser beim Lesen von u zunächst zum Zustand q. Beim Lesen von v läuft er im Kreis wieder zu q zurück. Wie oft dieser Kreis durchlaufen wird, spielt für den DFA keine Rolle, denn anschließend läuft er beim Lesen von w immer zu demselben Zustand $q' \in F$. □

Wir wenden das Pumping-Lemma auf zwei Beispiele an. Zunächst betrachten wir wieder die Sprache $L = \{0^i1^i \mid i \in \mathbb{N}\}$. Wie nehmen an, dass es einen DFA für diese Sprache L gibt. Dann können wir das Pumping-Lemma anwenden. Es sei n die Konstante mit den dort genannten Eigenschaften. Wir betrachten das Wort $z = 0^n1^n \in L$ der Länge $|z| = 2n \geq n$. Sei $z = uvw$ mit $|uv| \leq n$ und $|v| \geq 1$ eine beliebige Zerlegung dieses Wortes. Wir betrachten nun das Wort uv^2w. Da $|uv| \leq n$ ist und z mit n Nullen beginnt, gilt $v = 0^{|v|}$. Demzufolge gilt $uv^2w = 0^{n+|v|}1^n$. Wegen $|v| \geq 1$ gehört dieses Wort nicht zu L. Dies ist ein Widerspruch zu der Aussage des Pumping-Lemmas. Demzufolge ist die Annahme falsch, dass es einen DFA für L gibt.

Nun betrachten wir noch die Sprache $L = \{w \in \{0,1\}^* \mid w = w^R\}$ aller Palindrome. Angenommen es gibt einen DFA für L. Sei dann n die Konstante aus dem Pumping-Lemma und sei $z = 0^n10^n \in L$. Sei nun $z = uvw$ mit $|uv| \leq n$ und $|v| \geq 1$ eine beliebige Zerlegung dieses Wortes. Wir betrachten nun das Wort uv^2w. Da $|uv| \leq n$ ist und z mit n Nullen beginnt, gilt $v = 0^{|v|}$. Demzufolge gilt $uv^2w = 0^{n+|v|}10^n$. Wegen $|v| \geq 1$ gehört dieses Wort nicht zu L. Somit ist gezeigt, dass es keinen DFA für L gibt.

3.2.2 Das Pumping-Lemma als Spiel

Da das Pumping-Lemma von Studierenden, die es zum ersten Mal sehen, oft als unübersichtlich empfunden wird, wollen wir es noch einmal in etwas anderer Gestalt darstellen. Wir können es mithilfe von Quantoren wie folgt ausdrücken.

$$\begin{aligned} &\text{Es gibt einen DFA für } L \subseteq \Sigma^*. \\ &\quad \Rightarrow \exists n \in \mathbb{N} : \\ &\qquad \forall z \in L, |z| \geq n : \\ &\qquad\quad \exists \text{ Zerlegung } z = uvw, |uv| \leq n, |v| \geq 1 : \\ &\qquad\qquad \forall i \geq 0 : uv^iw \in L \end{aligned}$$

Mit dieser Aussage zeigt man in der Regel, dass gewisse Sprachen von keinem DFA entschieden werden. Man zeigt also, dass die Aussage rechts vom Implikationspfeil

nicht gilt (dass also ihre Negation gilt) und folgert daraus, dass L nicht von einem DFA entschieden werden kann. Dieses Vorgehen entspricht Theorem 2.1 c). Wir sollten uns bei der Negation der Aussage auf der rechten Seite insbesondere an die Regeln zur Negation von Aussagen mit Quantoren aus Theorem 2.9 erinnern. Es ergibt sich die folgende Aussage.

$$
\begin{aligned}
&\forall n \in \mathbb{N}: \\
&\qquad \exists z \in L, |z| \geq n: \\
&\qquad\qquad \forall \text{ Zerlegung } z = uvw, |uv| \leq n, |v| \geq 1: \\
&\qquad\qquad\qquad \exists i \geq 0 : uv^i w \notin L: \\
&\Rightarrow \text{Es gibt keinen DFA für } L \subseteq \Sigma^*.
\end{aligned}
$$

Wie zeigt man nun aber, dass diese Negation des Pumping Lemmas gilt? Man kann sich dies als ein Spiel zweier Personen vorstellen. Peter (Prover) möchte Vera (Verifier) davon überzeugen, dass die Aussage gilt. Vera möchte sich aber nur davon überzeugen lassen, wenn die Aussage wirklich wahr ist. Vera übernimmt die $\forall$-Rolle und Peter die $\exists$-Rolle. Das Spiel sieht wie folgt aus.

1. Runde: Vera wählt ein $n \in \mathbb{N}$.

2. Runde: Peter wählt ein $z \in L$ mit $|z| \geq n$.

3. Runde: Vera wählt eine Zerlegung $z = uvw$ mit $|uv| \leq n$ und $|v| \geq 1$.

4. Runde: Peter wählt ein $i \geq 0$.

Gelingt es Peter, in der 4. Runde ein i so zu wählen, dass $uv^i w \notin L$ gilt, so hat er gewonnen. Hat er eine Strategie, auf jede Wahl, die Vera in den Runden 1 und 3 trifft, zu reagieren, und schafft es am Ende immer, ein i zu wählen, sodass $uv^i w \notin L$ gilt, so gilt die Negation des Pumping-Lemmas und die Sprache ist nicht regulär.

Wir betrachten wieder die Sprache $L = \{0^i 1^i \mid i \in \mathbb{N}\}$ als Beispiel. Wir wollen zeigen, dass L nicht regulär ist und übernehmen Peters Rolle.

1. Runde: Vera wählt ein beliebiges $n \in \mathbb{N}$.

2. Runde: Wir wählen $z = 0^n 1^n$. Offenbar gilt $z \in L$ und $|z| \geq n$.

3. Runde: Vera wählt eine beliebige Zerlegung $z = uvw = 0^n 1^n$, wobei $|uv| \leq n$ und $|v| \geq 1$ gilt.

4. Runde: Da uv höchstens die Länge n haben darf und das Wort z mit n Nullen beginnt, gilt $uv = 0^j$ für ein $j \leq n$. Ebenso folgt $v = 0^k$ für ein $k \geq 1$. Wir wählen nun $i = 2$.

Mit unseren Vorüberlegungen gilt $uv^2 w = 0^{j-k} 0^{2k} 0^{n-j} 1^n = 0^{n+k} 1^n$. Wegen $k \geq 1$ gilt $0^{n+k} 1^n \notin L$. Wir haben uns also auf ein beliebiges n und eine beliebige Zerlegung eingestellt und in jedem Fall ein Wort $uv^i w$ gefunden, das nicht zur Sprache gehört. Also haben wir gewonnen und gezeigt, dass die Sprache nicht regulär ist.

3.2.3 Nichtdeterministische endliche Automaten

Ein in der theoretischen Informatik zentraler Begriff ist der des *Nichtdeterminismus*. Wir werden diesen Begriff in der Vorlesung „Algorithmen und Berechnungskomplexität II“ ausführlich behandeln und deshalb ist es gut, sich bereits jetzt bei dem einfachen Modell der endlichen Automaten mit ihm vertraut zu machen.

Definition 3.9. *Ein* nichtdeterministischer endlicher Automat *(nondeterministic finite automaton, NFA) M besteht aus fünf Komponenten $(Q, \Sigma, \delta, q_0, F)$. Der einzige Unterschied zu einem DFA ist die Zustandsüberführungsfunktion, die bei einem NFA eine* Zustandsüberführungsrelation *ist. Es gilt also $\delta \subseteq (Q \times \Sigma) \times Q$.*

Wir können δ ebenso als eine Abbildung von $Q \times \Sigma$ in die Potenzmenge von Q auffassen. Dann ist $\delta(q, a)$ die Menge aller Zustände q', für die $((q, a), q') \in \delta$ gilt. Wenn der Automat im Zustand q das Zeichen a liest, so ist der Nachfolgezustand q' nicht mehr eindeutig bestimmt, sondern es gibt stattdessen eine Menge von möglichen Nachfolgezuständen, die erreicht werden können. Genau wie bei DFAs erweitern wir diese Abbildung von $Q \times \Sigma$ in die Potenzmenge von Q zu einer Abbildung δ^* von $Q \times \Sigma^*$ in die Potenzmenge von Q. Dabei sei für alle $q \in Q$ und $a \in \Sigma$

$$\delta^*(q, \varepsilon) = \{q\} \quad \text{und} \quad \delta^*(q, a) = \delta(q, a).$$

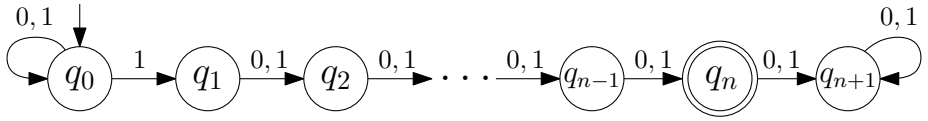
Außerdem definieren wir für jedes Wort $w = w_1 \dots w_n \in \Sigma^*$ der Länge $n \geq 2$

$$\delta^*(q, w) = \bigcup_{p \in \delta(q, w_1)} \delta^*(p, w_2 \dots w_n).$$

Erhält der NFA nun ein Wort $w \in \Sigma^*$ als Eingabe, so ist $\delta^*(q_0, w)$ die Menge der Zustände, die er erreichen kann. Falls $\delta^*(q_0, w) \cap F \neq \emptyset$ gilt, falls also die Möglichkeit besteht, dass der NFA nach dem Lesen von w einen akzeptierenden Zustand erreicht, so sagen wir, dass der NFA, das Wort w *akzeptiert*. Ansonsten sagen wir, dass der NFA das Wort w *verwirft*. Mit $L(M) \subseteq \Sigma^*$ bezeichnen wir für einen NFA M die Menge aller Wörter, die M akzeptiert. Wir sagen, dass M die Sprache $L(M)$ *entscheidet* oder *akzeptiert*. Wir sagen auch, dass M ein NFA für die Sprache $L(M)$ ist.

Wir werden gleich sehen, dass es eine Sprache gibt, für die es einen NFA gibt, der sehr viel weniger Zustände besitzt als jeder DFA für diese Sprache. In diesem Sinne können NFAs benutzt werden, um manche Sprachen kompakter zu beschreiben. Wir werden jedoch auch sehen, dass DFAs und NFAs in dem Sinne gleich mächtig sind, dass es zu jedem NFA einen DFA gibt, der dieselbe Sprache entscheidet (wenn auch mit mehr Zuständen). Wir werden NFAs hauptsächlich als theoretisches Hilfsmittel in einigen der folgenden Überlegungen einsetzen.

Auch NFAs können wir uns wieder als Übergangsgraphen vorstellen. Der einzige Unterschied zu DFAs ist, dass nun ein Knoten mehrere ausgehende Kanten haben kann, die mit demselben Zeichen aus Σ beschriftet sind. Betrachten wir das folgende Beispiel.



In diesem Beispiel ist der Zustand q_0 der einzige Zustand, bei dem vom Nichtdeterminismus Gebrauch gemacht wird. Wird in diesem Zustand eine Eins gelesen, so kann sich der NFA entscheiden, ob er im Zustand q_0 bleibt, oder ob er in den Zustand q_1 wechselt. Alle anderen Zustandsübergänge sind deterministisch. Die Sprache, die dieser NFA entscheidet, ist

$$L = \{w \in \{0,1\}^* \mid \text{das } n\text{-letzte Zeichen von } w \text{ ist } 1\}.$$

Um dies zu zeigen, sei zunächst w ein Wort, dessen n-letztes Zeichen 1 ist. Dieses Wort können wir als $w = u1v$ mit $u \in \{0,1\}^*$ und $v \in \{0,1\}^{n-1}$ schreiben. Der NFA kann dann beim Lesen von u im Zustand q_0 bleiben und danach beim Lesen der Eins in den Zustand q_1 wechseln. Dann wechselt er anschließend deterministisch beim Lesen von v in den akzeptierenden Zustand q_n. Somit akzeptiert der NFA das Wort w, da er den Zustand q_n erreichen kann. Nun müssen wir noch zeigen, dass er kein Wort $w \in \{0,1\}^*$ akzeptiert, dessen n-letztes Zeichen eine Null ist. Sei $w = u0v$ mit $u \in \{0,1\}^*$ und $v \in \{0,1\}^{n-1}$. Um nach dem Lesen von w im einzigen akzeptierenden Zustand q_n zu sein, muss der NFA genau beim Lesen des n-letzten Zeichens vom Zustand q_0 in den Zustand q_1 wechseln. Wechselt er früher von q_0 nach q_1, so erreicht er den Zustand q_{n+1}, wechselt er später oder gar nicht, so erreicht er einen Zustand q_i mit $i < n$. Der Wechsel von q_0 nach q_1 beim Lesen des n-letzten Zeichens ist aber nicht möglich, da dies eine Null ist. Somit gibt es keine Möglichkeit für den NFA nach dem Lesen von w in einem akzeptierenden Zustand zu sein.

Bei der Konstruktion eines DFA für L besteht die Schwierigkeit darin, dass wir beim Lesen eines Zeichens noch nicht wissen, ob es das n-letzte Zeichen ist. Beim NFA konnten wir dieses Problem einfach dadurch umgehen, dass wir dem NFA zwei mögliche Zustandsübergänge gegeben haben, wenn er im Startzustand eine Eins liest. Er hat dann gewissermaßen die Möglichkeit, zu raten, ob es sich bei der gerade gelesenen Eins um das n-letzte Zeichen handelt. Rät er richtig, so erreicht er den akzeptierenden Zustand. Tatsächlich können wir zeigen, dass jeder DFA für die Sprache L deutlich mehr Zustände haben muss.

Theorem 3.10. *Jeder DFA für die oben definierte Sprache L hat mindestens 2^n Zustände.*

Beweis. Es sei M ein DFA für die Sprache L. Es gibt 2^n Wörter in der Menge $\{0,1\}^n$ und wir behaupten, dass sich der DFA M nach dem Lesen zweier verschiedener Wörter $x, y \in \{0,1\}^n$ auch in verschiedenen Zuständen befinden muss. Damit folgt direkt, dass M mindestens 2^n Zustände besitzt.

Seien also $x = x_1 \dots x_n \in \{0,1\}^n$ und $y = y_1 \dots y_n \in \{0,1\}^n$ mit $x \neq y$ zwei beliebige Wörter aus $\{0,1\}^n$. Wir nehmen an, dass sich M nach dem Lesen von x und y in demselben Zustand q befindet. Wegen $x \neq y$ gibt es einen Index $i \in \{1, \dots, n\}$ mit $x_i \neq y_i$. Es gelte ohne Beschränkung der Allgemeinheit $x_i = 0$ und $y_i = 1$. Wir betrachten nun die Wörter $x0^{i-1}$ und $y0^{i-1}$. Das n-letzte Zeichen dieser Wörter ist x_i bzw. y_i. Also gilt $x0^{i-1} \notin L$ und $y0^{i-1} \in L$.

Da der DFA M die Sprache L entscheidet, gilt

$$\delta^*(q_0, y0^{i-1}) = \delta^*(q, 0^{i-1}) \in F.$$

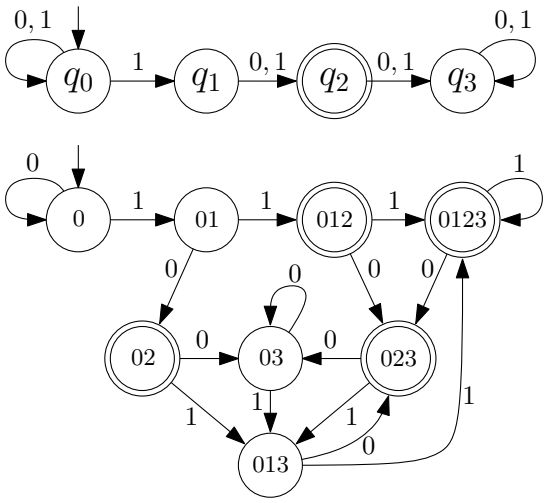


Abbildung 3.1: Beispiel zu Theorem 3.11: Oben ist ein NFA gezeigt, der genau die Wörter aus $\{0,1\}^*$ akzeptiert, deren vorletztes Zeichen eine Eins ist (siehe auch das Beispiel auf Seite 43). Unten ist der dazugehörige DFA gezeigt, der aus der Potenzmengenkonstruktion in Theorem 3.11 hervorgeht. Dabei haben wir aus Platzgründen eine abkürzende Schreibweise für die Zustände benutzt. Formal korrekt müsste es statt 0 und 01 beispielsweise $\{q_0\}$ und $\{q_0, q_1\}$ heißen. Außerdem haben wir Zustände, die vom Startzustand nicht erreicht werden können, weggelassen.

Unsere Annahme, dass sich der DFA M nach dem Lesen von x ebenfalls in Zustand q befindet, impliziert

$$\delta^*(q_0, x0^{i-1}) = \delta^*(q, 0^{i-1}) \in F.$$

Demzufolge akzeptiert der DFA M das Wort $x0^{i-1}$, das nicht zu der Sprache L gehört. Dies ist ein Widerspruch zu der Annahme, dass M die Sprache L entscheidet. Dies beweist, wie gewünscht, dass M sich nach dem Lesen zweier verschiedener Wörter $x, y \in \{0,1\}^n$ in verschiedenen Zuständen befinden muss. □

Es stellt sich nun sofort die Frage, ob es eine Sprache gibt, die von einem NFA entschieden wird, für die es aber keinen DFA gibt. Das folgende Theorem zeigt, dass dies nicht der Fall ist.

Theorem 3.11. *Zu jedem NFA mit n Zuständen gibt es einen DFA mit 2^n Zuständen, der dieselbe Sprache entscheidet.*

Beweis. Es sei $M = (Q, \Sigma, \delta, q_0, F)$ ein beliebiger NFA mit $n = |Q|$. Wir konstruieren einen DFA $M' = (Q', \Sigma, \delta', q_0', F')$ mit $L(M) = L(M')$ und $|Q'| = 2^n$. Für diesen DFA gilt:

- Q' ist die Potenzmenge von Q,
- $q_0' = \{q_0\}$,
- $F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$,
- $\delta' : Q' \times \Sigma \to Q'$ ist für $q \in Q'$ und $a \in \Sigma$ definiert als

$$\delta'(q, a) = \bigcup_{p \in q} \delta(p, a).$$

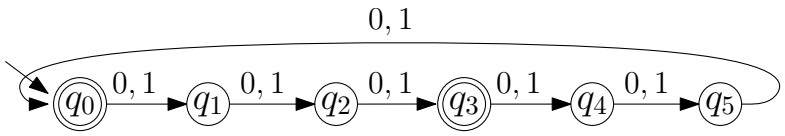


Abbildung 3.2: Beispiel für einen nicht-minimalen DFA

Diese Konstruktion wird auch *Potenzmengenkonstruktion* genannt, und es gilt gemäß Theorem 2.8 wie gewünscht $|Q'| = 2^n$. Wir müssen nur noch zeigen, dass M' dieselbe Sprache wie M entscheidet. Dazu genügt es zu zeigen, dass die Menge $(\delta')^*(q, w)$ genau diejenigen Zustände aus Q enthält, die der NFA M beim Lesen des Eingabewortes w erreichen kann, wenn er von einem beliebigen Zustand $p \in q$ startet. Dies folgt unmittelbar aus der Definition von δ'. Formal kann es mit vollständiger Induktion über die Länge von w gezeigt werden. □

Abbildung 3.1 zeigt ein Beispiel für die Konstruktion aus Theorem 3.11. Damit haben wir gezeigt, dass jede Sprache, die von einem NFA entschieden werden kann, auch von einem DFA entschieden werden kann. Andersherum sind DFAs nur spezielle NFAs und daher kann auch jede Sprache, die von einem DFA entschieden wird, von einem NFA entschieden werden. Somit sind diese beiden Klassen von Sprachen identisch und DFAs und NFAs sind gleichmächtig.

3.2.4 Minimierung endlicher Automaten

Oft möchte man für eine gegebene Sprache L nicht nur einen beliebigen DFA finden, sondern einen DFA mit möglichst wenigen Zuständen. Wir werden in diesem Abschnitt sehen, wie man für einen beliebigen DFA M einen DFA M' konstruieren kann, für den $L(M) = L(M')$ gilt und der unter allen DFAs mit dieser Eigenschaft die kleinstmögliche Anzahl an Zuständen besitzt. Einen solchen DFA M' nennen wir auch *minimalen DFA für die Sprache* $L(M)$.

Abbildung 3.2 zeigt als Beispiel einen DFA, der genau diejenigen Wörter $w \in \{0,1\}^*$ akzeptiert, deren Länge ein Vielfaches von drei ist. Jeder Zustand dieses DFA kann bei einem entsprechenden Wort erreicht werden, aber dennoch kann man sich leicht davon überzeugen, dass es einen DFA mit nur drei Zuständen gibt, der dieselbe Sprache entscheidet. Ebenso kann man in diesem Beispiel relativ leicht zeigen, dass es keinen DFA mit weniger als drei Zuständen gibt, der dieselbe Sprache entscheidet.

Wir interessieren uns nun für ein allgemeines Verfahren, um DFAs zu minimieren. Dazu führen wir zunächst die folgende Definition ein.

Definition 3.12. *Zwei Zustände $p \in Q$ und $q \in Q$ eines DFA $M = (Q, \Sigma, \delta, q_0, F)$ heißen* äquivalent*, wenn für alle Wörter $w \in \Sigma^*$ gilt:*

$$\delta^*(p, w) \in F \iff \delta^*(q, w) \in F.$$

Wir schreiben dann $p \equiv q$. Mit $[\![p]\!] \subseteq Q$ bezeichnen wir die Menge aller Zustände, die äquivalent zu p sind.

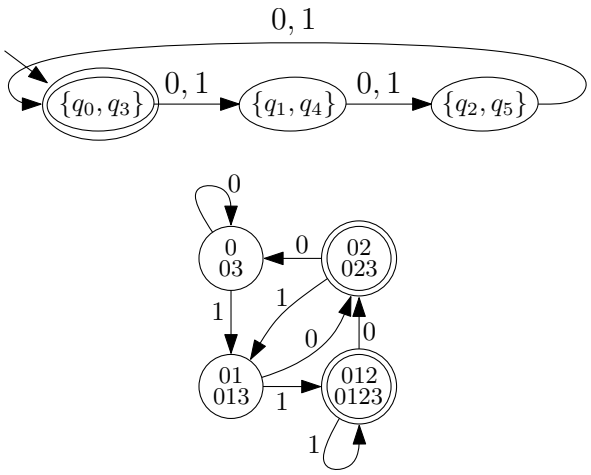


Abbildung 3.3: Äquivalenzklassenautomaten zu den DFAs aus Abbildung 3.2 (oben) und Abbildung 3.1 (unten)

Wenn zwei Zustände p und q äquivalent sind, dann spielt es für das Akzeptanzverhalten des DFA keine Rolle, ob er sich in Zustand p oder q befindet. Liest er startend in p oder q dasselbe Wort w, so akzeptiert er entweder in beiden Fällen oder er verwirft in beiden Fällen. Der Leser sollte sich davon überzeugen, dass $\equiv$ eine Äquivalenzrelation auf der Zustandsmenge Q ist. Das bedeutet insbesondere, dass die Relation $\equiv$ die Zustandsmenge Q in Äquivalenzklassen einteilt. In dem Beispiel in Abbildung 3.2 sind dies die Klassen $\{q_0, q_3\}$, $\{q_1, q_4\}$ und $\{q_2, q_5\}$. Wir konstruieren nun einen DFA, dessen Zustandsmenge der Menge der Äquivalenzklassen der Relation $\equiv$ entspricht.

Definition 3.13. *Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Der DFA $M' = (Q', \Sigma, \delta', q_0', F')$ mit den folgenden Komponenten heißt* Äquivalenzklassenautomat zu *M:*

- $Q' = \{[\![q]\!] \mid q \in Q\}$*: Die Zustände entsprechen den Äquivalenzklassen von $\equiv$.*
- Σ*: Der Äquivalenzklassenautomat arbeitet auf demselben Alphabet wie M.*
- $q_0' = [\![q_0]\!]$*: Der Startzustand ist die Äquivalenzklasse des Startzustandes.*
- $F' = \{[\![q]\!] \mid q \in F\}$*: Die akzeptierenden Zustände sind die Äquivalenzklassen der akzeptierenden Zustände von M.*
- $\delta' : Q' \times \Sigma \to Q'$ *mit* $\delta'([\![q]\!], a) = [\![\delta(q, a)]\!]$ *für alle* $q \in Q$ *und* $a \in \Sigma$.

Abbildung 3.3 zeigt zwei Beispiele für Äquivalenzklassenautomaten. Anhand der Definition ist nicht direkt klar, dass δ' wohldefiniert ist. Gäbe es zwei Zustände $p, q \in Q$ mit $[\![p]\!] = [\![q]\!]$ und ein $a \in \Sigma$ mit $[\![\delta(p, a)]\!] \neq [\![\delta(q, a)]\!]$, so wäre die obige Definition widersprüchlich. Wir zeigen nun, dass dieser Fall nicht eintreten kann.

Theorem 3.14. *Der Äquivalenzklassenautomat M' ist wohldefiniert und entscheidet dieselbe Sprache wie M.*

Beweis. Zunächst zeigen wir, dass δ' wohldefiniert ist. Dazu genügt es zu zeigen, dass für $p, q \in Q$ mit $[\![p]\!] = [\![q]\!]$ und $a \in \Sigma$ stets $[\![\delta(p,a)]\!] = [\![\delta(q,a)]\!]$ gilt. Dies folgt aus der Definition von $\equiv$, wie das folgende Argument zeigt:

$$\begin{aligned}
[\![p]\!] = [\![q]\!] &\Rightarrow p \equiv q \\
&\Rightarrow \forall w \in \Sigma^* : && [\delta^*(p,w) \in F \iff \delta^*(q,w) \in F] \\
&\Rightarrow \forall a \in \Sigma : \forall w \in \Sigma^* : && [\delta^*(p,aw) \in F \iff \delta^*(q,aw) \in F] \\
&\Rightarrow \forall a \in \Sigma : \forall w \in \Sigma^* : && [\delta^*(\delta(p,a),w) \in F \iff \delta^*(\delta(q,a),w) \in F] \\
&\Rightarrow \forall a \in \Sigma : && \delta(p,a) \equiv \delta(q,a) \\
&\Rightarrow \forall a \in \Sigma : && [\![\delta(p,a)]\!] = [\![\delta(q,a)]\!].
\end{aligned}$$

Es bleibt noch zu zeigen, dass M' dieselbe Sprache wie M akzeptiert. Als erstes beobachten wir die folgende Eigenschaft: Gilt $[\![p]\!] = [\![q]\!]$ für zwei Zustände $p, q \in Q$, so sind entweder p und q beide akzeptierende Zustände oder p und q sind beide keine akzeptierenden Zustände. Dies folgt direkt aus der Definition von $\equiv$, denn $p \equiv q$ impliziert, dass für alle $w \in \Sigma^*$ die Äquivalenz $\delta^*(p,w) \in F \iff \delta^*(q,w) \in F$ gilt. Wir können insbesondere $w = \varepsilon$ einsetzen und erhalten

$$p = \delta^*(p,\varepsilon) \in F \iff q = \delta^*(q,\varepsilon) \in F.$$

Sei nun $w \in \Sigma^*$ und sei $q_0, q_1, \ldots, q_n$ die Zustandsfolge, die der DFA M beim Lesen von w durchläuft. Gemäß der Definition des Äquivalenzklassenautomaten durchläuft M' die Zustandsfolge $[\![q_0]\!], [\![q_1]\!], \ldots, [\![q_n]\!]$. Der DFA M akzeptiert w genau dann, wenn $q_n \in F$ gilt. Der DFA M' akzeptiert w genau dann, wenn $[\![q_n]\!] \in F'$ gilt. Nach Definition von F' gilt $[\![q_n]\!] \in F'$ genau dann, wenn es einen Zustand $q \in F$ mit $q \equiv q_n$ gibt. Gemäß der obigen Beobachtung ist dies genau dann der Fall, wenn $q_n \in F$ gilt. Somit akzeptieren entweder M und M' beide das Wort w oder beide verwerfen es. □

Wenn wir im Folgenden über den Äquivalenzklassenautomaten M' zu einem gegebenen DFA M sprechen, so gehen wir davon aus, dass wir aus der Zustandsmenge Q' alle *überflüssigen* Zustände entfernt haben. Das sind die Zustände, die vom Startzustand aus nicht erreicht werden können. Formal heißt ein Zustand $q' \in Q'$ überflüssig, wenn es kein $w \in \Sigma^*$ mit $(\delta')^*(q'_0, w) = q'$ gibt.

Theorem 3.15. *Der Äquivalenzklassenautomat M' zu einem DFA M ist ein minimaler DFA für die Sprache $L(M)$*

Um dieses Theorem zu beweisen, führen wir zunächst als Hilfsmittel die *Nerode-Relation* ein, die nach dem Mathematiker Anil Nerode benannt ist.

Definition 3.16. *Sei $L \subseteq \Sigma^*$ eine Sprache. Zwei Wörter $x, y \in \Sigma^*$ stehen bezüglich der* Nerode-Relation *R_L genau dann in Beziehung (d. h. $x\, R_L\, y$), wenn gilt:*

$$\forall z \in \Sigma^* : xz \in L \iff yz \in L.$$

Es kann leicht überprüft werden, dass es sich bei der Nerode-Relation um eine Äquivalenzrelation handelt, die die Menge Σ^* in Äquivalenzklassen einteilt. Die Anzahl der Äquivalenzklassen von R_L nennen wir den *Index von R_L* und schreiben dafür $\text{index}(R_L)$. Im Allgemeinen kann der Index von R_L auch unendlich sein.

Lemma 3.17. *Die Anzahl der Zustände eines beliebigen DFA für eine Sprache L beträgt mindestens* $\operatorname{index}(R_L)$.

Beweis. Sei $\ell = \operatorname{index}(R_L)$ und seien $w_1, \ldots, w_\ell \in \Sigma^*$ Wörter, die paarweise nicht in Relation bezüglich R_L stehen. Ferner sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA für die Sprache L. Wir nehmen an, dass $|Q| < \ell$ gilt. Dann gibt es zwei Wörter w_i und w_j mit $i \neq j$, für die $\delta^*(q_0, w_i) = \delta^*(q_0, w_j)$ gilt. Da w_i und w_j nicht in Relation bezüglich R_L stehen, gibt es ein $z \in \Sigma^*$ mit $w_i z \in L \not\iff w_j z \in L$. Sei ohne Beschränkung der Allgemeinheit $w_i z \in L$ und $w_j z \notin L$. Da M die Sprache L entscheidet, gilt $\delta^*(q_0, w_i z) \in F$. Damit folgt auch

$$\delta^*(q_0, w_j z) = \delta^*(\delta^*(q_0, w_j), z) = \delta^*(\delta^*(q_0, w_i), z) = \delta^*(q_0, w_i z) \in F.$$

Somit akzeptiert der DFA M das Wort $w_j z$ im Widerspruch dazu, dass er die Sprache L entscheidet. □

Lemma 3.18. *Sei $M' = (Q', \Sigma, \delta', q_0', F')$ der Äquivalenzklassenautomat zu einem DFA $M = (Q, \Sigma, \delta, q_0, F)$. Ferner sei $L = L(M) = L(M')$. Dann gilt $|Q'| \leq \operatorname{index}(R_L)$.*

Beweis. Sei $k = |Q'|$ und seien $q_0, \ldots, q_{k-1} \in Q$ Repräsentanten der Äquivalenzklassen der Relation $\equiv$. Es gelte also $Q' = \{[\![q_0]\!], \ldots, [\![q_{k-1}]\!]\}$. Da wir davon ausgehen, dass M' keine überflüssigen Zustände enthält, gibt es für jeden Zustand $[\![q_i]\!] \in Q'$ ein Wort $w_i \in \Sigma^*$ mit $(\delta')^*([\![q_0]\!], w_i) = [\![q_i]\!]$. Nehmen wir an, dass es zwei Wörter w_i und w_j mit $i \neq j$ gibt, die in Relation bezüglich R_L stehen. Dann folgt aus der Definition der Nerode-Relation R_L

$$\forall z \in \Sigma^* : w_i z \in L \iff w_j z \in L.$$

Da der Äquivalenzklassenautomat M' die Sprache L entscheidet, impliziert dies

$$\forall z \in \Sigma^* : (\delta')^*([\![q_0]\!], w_i z) \in F' \iff (\delta')^*([\![q_0]\!], w_j z) \in F',$$

woraus

$$\forall z \in \Sigma^* : (\delta')^*([\![q_i]\!], z) \in F' \iff (\delta')^*([\![q_j]\!], z) \in F'$$

folgt. Dies wiederum impliziert

$$\forall z \in \Sigma^* : \delta^*(q_i, z) \in F \iff \delta^*(q_j, z) \in F.$$

Demzufolge gilt $q_i \equiv q_j$ im Widerspruch dazu, dass q_i und q_j verschiedene Äquivalenzklassen von $\equiv$ repräsentieren.

Insgesamt folgt, dass die Wörter $w_0, \ldots, w_{k-1}$ in verschiedenen Äquivalenzklassen der Nerode-Relation liegen müssen. Somit gilt $\operatorname{index}(R_L) \geq k = |Q'|$. □

Beweis von Theorem 3.15. Das Theorem folgt direkt aus den vorangegangenen Lemmata. Laut Lemma 3.18 beträgt die Anzahl an Zuständen des Äquivalenzklassenautomaten höchstens $\operatorname{index}(R_{L(M)})$. Laut Lemma 3.17 gibt es keinen DFA für die Sprache $L(M)$, der mit weniger als $\operatorname{index}(R_{L(M)})$ Zuständen auskommt. □

Aus dem Theorem folgt auch eine Möglichkeit, die Menge der Sprachen zu charakterisieren, die von DFAs entschieden werden können. Es sind genau nämlich genau die Sprachen, deren zugehörige Nerode-Relation einen endlichen Index besitzt.

Es stellt sich nun noch die Frage, wie man den Äquivalenzklassenautomaten zu einem gegebenen DFA effizient bestimmen kann. Dies ist einfach, wenn die Äquivalenzklassen der Relation $\equiv$ bekannt sind. Es ist jedoch nicht offensichtlich, wie diese bestimmt werden können. Für zwei Zustände p und q ist nämlich nicht klar, wie getestet werden kann, ob $p \equiv q$ gilt. Da Σ^* unendlich viele Wörter enthält, kann man nicht für alle Wörter testen, ob $\delta^*(p,w) \in F \iff \delta^*(q,w) \in F$ gilt. Man kann jedoch zeigen, dass es im Falle $p \not\equiv q$ stets ein $w \in \Sigma^*$ der Länge höchstens $|Q|$ mit $\delta^*(p,w) \in F \not\Longleftrightarrow \delta^*(q,w) \in F$ gibt. Diese Beobachtung führt zumindest zu einem Algorithmus zur Berechnung der Äquivalenzklassen von $\equiv$ mit endlicher Laufzeit. Die Laufzeit ist allerdings sehr groß, da pro Zustandspaar bis zu $|\Sigma|^{|Q|}$ Wörter getestet werden müssen. Es gibt jedoch Algorithmen, die den Äquivalenzklassenautomaten sehr viel effizienter berechnen. Auf diese Algorithmen werden wir in dieser Vorlesung nicht weiter eingehen, der interessierte Leser findet aber zum Beispiel in dem Buch von Norbert Blum [1] die Beschreibung eines solchen Algorithmus.

3.3 Reguläre Sprachen, endliche Automaten und reguläre Ausdrücke

Wir kommen jetzt zum Beginn dieses Kapitels zurück und stellen den Zusammenhang zwischen regulären Sprachen und endlichen Automaten her. Einer der wesentlichen Gründe, warum wir uns mit Automaten beschäftigt haben, ist das folgende Theorem.

Theorem 3.19. *Die Klasse der Sprachen, die von DFAs entschieden werden können, stimmt mit der Klasse der regulären Sprachen überein.*

Beweis. Zuerst zeigen wir, wie man für einen DFA $M = (Q, \Sigma, \delta, q_0, F)$ eine reguläre Grammatik $G = (\Sigma, V, S, P)$ konstruieren kann, für die $L(G) = L(M)$ gilt. Die Idee besteht darin, mithilfe der regulären Grammatik die Berechnung des DFA zu simulieren. Dazu setzen wir $V = Q$ und $S = q_0$. Die Menge P enthält für alle $q, q' \in Q$ und $a \in \Sigma$ mit $\delta(q,a) = q'$ eine Regel $q \to aq'$. Des Weiteren gibt es in P für alle $q \in F$ die Regel $q \to \varepsilon$.

Sei $w = w_1 \dots w_n \in L(M)$ ein Wort, das der DFA M akzeptiert, und sei $q_0, q_1, \dots, q_n$ die Zustandsfolge, die M beim Lesen von w durchläuft. Dann gilt $\delta(q_{i-1}, w_i) = q_i$ für alle $i \geq 1$ und $q_n \in F$. Demzufolge enthält P die Ableitungsregeln $q_{i-1} \to w_i q_i$ sowie $q_n \to \varepsilon$ und wir können das Wort w wie folgt ableiten:

$$q_0 \to w_1 q_1 \to w_1 w_2 q_2 \to \dots \to w_1 \dots w_n q_n \to w_1 \dots w_n \varepsilon = w.$$

Es gilt also $w \in L(G)$. Daraus folgt $L(M) \subseteq L(G)$.

Sei umgekehrt $w = w_1 \dots w_n \in L(G)$ ein Wort, das wir mit der Grammatik ableiten können. Wegen der eingeschränkten Regeln der Grammatik muss die Ableitung von w

wieder die obige Form für eine Zustandsfolge $q_0, \ldots, q_n$ mit $q_n \in F$ haben. Weiterhin muss für diese Zustandsfolge wegen der Definition von P wieder $q_{i-1} \to w_i q_i$ für alle $i \geq 1$ gelten. Demzufolge gilt $\delta(q_{i-1}, w_i) = q_i$ für alle $i \geq 1$ und somit akzeptiert auch der DFA M das Wort w. Daraus folgt $L(G) \subseteq L(M)$. Insgesamt haben wir gezeigt, dass $L(G) = L(M)$ gilt.

Nun zeigen wir, wie man für eine reguläre Grammatik $G = (\Sigma, V, S, P)$ einen DFA konstruieren kann, der die Sprache $L(G)$ entscheidet. Nach Theorem 3.11 genügt es, einen NFA $M = (Q, \Sigma, \delta, q_0, F)$ mit $L(M) = L(G)$ zu konstruieren. Wir setzen $Q = V$, $q_0 = S$ und $F = \{A \in V \mid (A \to \varepsilon) \in P\}$. Außerdem sei

$$\delta(A, a) = \{B \in V \mid (A \to aB) \in P\}.$$

Sei $w = w_1 \ldots w_n \in L(G)$. Dann gibt es eine Ableitung

$$S \to w_1 A_1 \to w_1 w_2 A_2 \to \cdots \to w_1 \ldots w_n A_n \to w_1 \ldots w_n \varepsilon = w. \tag{3.1}$$

Es gibt also die Regeln $S \to w_1 A_1$ und $A_{i-1} \to w_i A_i$ für $i \geq 2$ in P. Des Weiteren gibt es die Regel $A_n \to \varepsilon$ in P. Dementsprechend gilt $A_1 \in \delta(S, w_1)$ und $A_i \in \delta(A_{i-1}, w_i)$ für $i \geq 2$. Erhält der NFA M das Wort w als Eingabe, so kann er demzufolge die Zustandsfolge $S, A_1, \ldots, A_n$ durchlaufen. Wegen $(A_n \to \varepsilon) \in P$ ist $A_n \in F$. Somit akzeptiert der NFA M das Wort w. Daraus folgt $L(G) \subseteq L(M)$.

Sei umgekehrt $w = w_1 \ldots w_n \in L(M)$ ein Wort, das der NFA M akzeptiert. Dann gibt es eine Folge $S, A_1, \ldots, A_n$ von Zuständen mit $A_n \in F$ und mit $A_1 \in \delta(S, w_1)$ und $A_i \in \delta(A_{i-1}, w_i)$ für $i \geq 2$. Dementsprechend muss es in P die Ableitungsregeln $S \to w_1 A_1$, $A_{i-1} \to w_i A_i$ für $i \geq 2$ und $A_n \to \varepsilon$ geben. Damit gilt auch $w \in L(G)$, denn w kann gemäß der Ableitung (3.1) aus S abgeleitet werden. Daraus folgt $L(M) \subseteq L(G)$ und somit insgesamt $L(M) = L(G)$. □

Es gibt noch eine weitere gängige Möglichkeit, die Klasse der regulären Sprachen zu charakterisieren, nämlich mithilfe sogenannter *regulärer Ausdrücke*.

Definition 3.20. *Sei Σ ein endliches Alphabet. Die Menge der* regulären Ausdrücke *über Σ ist die Menge der Ausdrücke, die sich durch endlich viele Anwendungen der folgenden beiden Regeln erzeugen lassen. Dabei repräsentiert jeder reguläre Ausdruck R eine Sprache $L(R) \subseteq \Sigma^*$.*

a) Die Ausdrücke $\emptyset$, ε und a für $a \in \Sigma$ sind reguläre Ausdrücke. Dabei ist $\emptyset$ ein regulärer Ausdruck, der die leere Sprache $\emptyset \subseteq \Sigma^*$ beschreibt, ε ist ein regulärer Ausdruck, der die Sprache $\{\varepsilon\}$ beschreibt, und a ist ein regulärer Ausdruck, der die Sprache $\{a\}$ beschreibt. Es gilt also $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ und $L(a) = \{a\}$.

b) Sind R_1 und R_2 reguläre Ausdrücke, die die Sprachen $L_1 = L(R_1)$ und $L_2 = L(R_2)$ beschreiben, dann gilt:

- *$(R_1) + (R_2)$ ist ein regulärer Ausdruck für die Sprache $L((R_1) + (R_2)) = L_1 \cup L_2$.*

- $(R_1) \cdot (R_2)$ *ist ein regulärer Ausdruck für die Sprache* $L((R_1) \cdot (R_2)) = L_1 \cdot L_2 = \{w_1 w_2 \in \Sigma^* \mid w_1 \in L_1, w_2 \in L_2\}$*, die sogenannte* Konkatenation *von* L_1 *und* L_2*.*
- $(R_1)^*$ *ist ein regulärer Ausdruck für den* Kleeneschen Abschluss L_1^* *von* L_1*. Dabei handelt es sich um die Sprache* $L((R_1)^*) = L_1^* = \cup_{i \geq 0} L_1^i$ *mit* $L_1^0 = \{\varepsilon\}$ *und* $L_1^i = L_1 \cdot L_1^{i-1}$ *für* $i \geq 1$*. Der Kleenesche Abschluss* L_1^* *enthält also alle endlichen Konkatenationen von Wörtern aus* L_1*.*

Zur Verdeutlichung der Konkatenation und des Kleeneschen Abschlusses betrachten wir zunächst einige Beispiele unabhängig von regulären Ausdrücken.

- Für $L_1 = \{\varepsilon, 111\}$ und $L_2 = \{\varepsilon, 01\}$ gilt

$$L_1 \cdot L_2 = \{\varepsilon, 01, 111, 11101\} \quad \text{und} \quad L_2 \cdot L_1 = \{\varepsilon, 111, 01, 01111\}.$$

- Für $L_1 = \{0^n 1^n \mid n \in \mathbb{N}_0\}$ und $L_2 = \{1^n 0^n \mid n \in \mathbb{N}_0\}$ gilt

$$L_1 \cdot L_2 = \{0^n 1^{n+m} 0^m \mid n, m \in \mathbb{N}_0\}.$$

- Für $L = \{1, 00\}$ gilt $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = \{11, 100, 001, 0000\}$ und

$$L^3 = \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\}.$$

  Der Kleenesche Abschluss L^* von L enthält alle Wörter, die aus einer endlichen Aneinanderreihung der Wörter 1 und 00 in einer beliebigen Reihenfolge bestehen.

In regulären Ausdrücken lassen wir oft die Klammern um die elementaren Ausdrücke $\emptyset$, ε und a weg und wir definieren, dass * eine höhere Priorität als $\cdot$ hat, was wiederum eine höhere Priorität als $+$ hat. Außerdem lassen wir oft das Zeichen $\cdot$ weg. Wir schreiben also zum Beispiel 01 statt $(0) \cdot (1)$. Wir betrachten nun einige Beispiele für reguläre Ausdrücke.

- Es gilt
$$L((01)^*) = \{(01)^i \mid i \in \mathbb{N}_0\} = \{\varepsilon, 01, 0101, 010101, \ldots\}.$$
- Es gilt
$$L((0+1)^*) = \{0, 1\}^*.$$
- Außerdem ist
$$0(01)^* + (01 + 10)^*$$
  ein regulärer Ausdruck für die Sprache aller Wörter $w \in \{0, 1\}^*$, die entweder mit einer Null beginnen und anschließend beliebig viele Wiederholungen von 01 enthalten oder die aus beliebig vielen Wiederholungen der Blöcke 01 und 10 in einer beliebigen Reihenfolge bestehen.
- Wir können auch einen regulären Ausdruck für die Sprache angeben, die aus allen Wörtern besteht, die abwechselnd Nullen und Einsen enthalten:
$$(\varepsilon + 1)(01)^*(\varepsilon + 0).$$

Ist R ein regulärer Ausdruck, so findet man in vielen Anwendungen auch den Ausdruck R^+. Dies ist eine Abkürzung für den Ausdruck RR^*. Die Sprache, die dieser Ausdruck beschreibt, stimmt somit bis auf das fehlende leere Wort mit der Sprache des Ausdrucks R^* überein.

Leser, die mit dem Unix-Programm `grep` vertraut sind, haben reguläre Ausdrücke bereits gesehen. `grep` kann man als Parameter nämlich einen regulären Ausdruck übergeben und es sucht dann in Dateien nach Wörtern, die zu der Sprache dieses regulären Ausdrucks gehören.

Wir zeigen nun in zwei Schritten, dass die Klasse der Sprachen, die durch reguläre Ausdrücke beschrieben werden können, mit der Klasse der regulären Sprachen übereinstimmt.

Lemma 3.21. *Jeder reguläre Ausdruck R beschreibt eine reguläre Sprache $L(R)$.*

Beweis. Es sei ein beliebiger regulärer Ausdruck R gegeben. Wir zeigen, dass $L(R)$ regulär ist. Dabei geben wir die Struktur des Beweises vor und überlassen die Ausführung einiger Details dem Leser als Übung. Wir führen den Beweis mittels *struktureller Induktion*. Das ist ein Beweisprinzip, das dem Prinzip der vollständigen Induktion stark ähnelt. Der einzige Unterschied ist, dass man mithilfe von struktureller Induktion die Gültigkeit einer Menge von Aussagen nachweist, die mit einer rekursiv definierten Struktur parametrisiert sind. Bei einer vollständigen Induktion weist man hingegen die Gültigkeit einer Menge von Aussagen nach, die mit einer natürlichen Zahl parametrisiert sind.

Um dieses Prinzip zu veranschaulichen, betrachten wir zunächst noch einmal die Aussage, die wir nachweisen möchten. Dazu bezeichnen wir für einen regulären Ausdruck R mit $A(R)$ die Aussage, dass $L(R)$ regulär ist. Wir möchten nachweisen, dass die Aussage $A(R)$ für jeden regulären Ausdruck R gilt. Wir nutzen die Definition von regulären Ausdrücken und weisen im *Induktionsanfang* zunächst die Gültigkeit der Aussage $A(R)$ für alle nicht rekursiv definierten regulären Ausdrücke nach. Das sind die Ausdrücke $\emptyset$, ε und a für $a \in \Sigma$ aus Definition 3.20 a). Man sieht einfach, dass die Sprachen $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ und $L(a) = \{a\}$ regulär sind, da man für sie einen DFA oder eine reguläre Grammatik entwerfen kann.

Für den *Induktionsschritt* betrachten wir einen regulären Ausdruck R, der durch eine der drei erlaubten Regeln aus Definition 3.20 b) aus einem oder zwei regulären Ausdrücken hervorgeht. Sei also $R = (R_1) + (R_2)$, $R = (R_1) \cdot (R_2)$ oder $R = (R_1)^*$ für reguläre Ausdrücke R_1 und R_2. Wir gehen davon aus, dass die Aussagen $A(R_1)$ und $A(R_2)$ gelten und weisen unter dieser Annahme die Gültigkeit von $A(R)$ nach. Gelingen uns der Induktionsanfang und dieser Induktionsschritt, so besagt das Prinzip der strukturellen Induktion, dass wir die Gültigkeit von $A(R)$ für jeden regulären Ausdruck R nachgewiesen haben.

Wir führen nun den Induktionsschritt durch. Sei dazu ein beliebiger regulärer Ausdruck R der Form $R = (R_1) + (R_2)$, $R = (R_1) \cdot (R_2)$ oder $R = (R_1)^*$ gegeben. Wir gehen davon aus, dass die Sprachen $L_1 = L(R_1)$ und $L_2 = L(R_2)$ regulär sind. Das heißt für diese beiden Sprachen gibt es sowohl eine reguläre Grammatik als auch einen

DFA. Wir müssen zeigen, dass auch $L_1 \cup L_2$, $L_1 \cdot L_2$ und L_1^* regulär sind. Dies überlassen wir dem Leser als Übung. Gemäß der Theoreme 3.11 und 3.19 genügt es, für diese Sprachen einen DFA, NFA oder eine reguläre Grammatik zu entwerfen. Der Leser möge selbst ausprobieren, welches die einfachste Konstruktion ist.

Formal kann man die Korrektheit der strukturellen Induktion auf die Korrektheit der vollständigen Induktion zurückführen. Dazu führen wir für jedes $n \in \mathbb{N}_0$ zunächst die Menge $\mathcal{R}_n$ der regulären Ausdrücke ein, die durch die maximal n-fach verschachtelte Anwendung von Regeln aus Definition 3.20 b) aus den Grundausdrücken aus Definition 3.20 a) entstehen. Das bedeutet $\mathcal{R}_0$ enthält genau die regulären Ausdrücke aus Definition 3.20 a). Für $n \geq 1$ ist jeder reguläre Ausdruck $R \in \mathcal{R}_n$ entweder bereits in $\mathcal{R}_{n-1}$ enthalten oder er entsteht durch eine der drei Regeln aus Definition 3.20 b) aus einem oder zwei regulären Ausdrücken $R_1, R_2 \in \mathcal{R}_{n-1}$. Es gilt dann also $R = (R_1) + (R_2)$, $R = (R_1) \cdot (R_2)$ oder $R = (R_1)^*$.

Für $n \in \mathbb{N}_0$ führen wir nun die Aussage $B(n)$ ein, die besagt, dass die Aussage $A(R)$ für alle regulären Ausdrücke $R \in \mathcal{R}_n$ gilt. Im obigen Induktionsanfang wird die Aussage $B(0)$ nachgewiesen, da dort alle regulären Ausdrücke betrachtet werden, die ohne die Anwendung der Regeln aus Definition 3.20 b) entstehen. Im obigen Induktionsschritt wird die Gültigkeit von $B(n)$ unter der Annahme von $B(n-1)$ nachgewiesen. Es folgt somit aus dem Prinzip der vollständigen Induktion, dass die Aussage $B(n)$ für alle $n \in \mathbb{N}_0$ gilt. Damit ist bewiesen, dass $L(R)$ für jeden regulären Ausdruck R regulär ist, denn für jeden regulären Ausdruck R gibt es ein $n \in \mathbb{N}_0$ mit $R \in \mathcal{R}_n$. □

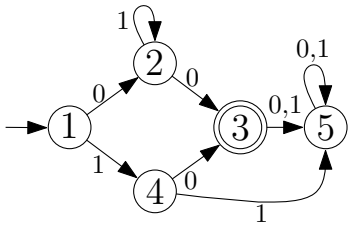
Lemma 3.22. *Für jede reguläre Sprache L gibt es einen regulären Ausdruck R mit $L = L(R)$.*

Beweis. Sei L eine beliebige reguläre Sprache. Wir wissen aus Theorem 3.19, dass es einen DFA $M = (Q, \Sigma, \delta, q_0, F)$ gibt, der die Sprache L entscheidet. Mithilfe dieses DFA werden wir nun einen regulären Ausdruck R für die Sprache L entwerfen. Wir wenden dazu *dynamische Programmierung* an. Das ist eine wichtige Technik zum Entwurf von Algorithmen, die wir im dritten Semester ausführlich besprechen werden. Die wesentliche Idee dieser Technik ist es, das eigentlich zu lösende Problem in kleinere Teilprobleme zu zerlegen. Dann löst man die Teilprobleme und setzt aus der Lösung der Teilprobleme eine Lösung für das Gesamtproblem zusammen.

Sei $n = |Q|$ und ohne Beschränkung der Allgemeinheit sei $Q = \{1, 2, \ldots, n\}$ und $q_0 = 1$. Wir definieren nun für jedes Tripel $i, j \in Q$ und $k \in \{0, 1, \ldots, n\}$ eine Sprache $L_{i,j}^k \subseteq \Sigma^*$. Ein Wort $w \in \Sigma^*$ gehört genau dann zu $L_{i,j}^k$, wenn sich der DFA M startend in Zustand i nach dem Lesen des Wortes w in Zustand j befindet und beim Lesen des Wortes w ausschließlich Zustände $q \in Q$ mit $q \leq k$ durchläuft. Dabei zählen der erste Zustand i und der letzte Zustand j nicht, das heißt, es darf durchaus $i > k$ und $j > k$ gelten. Formal gehört ein Wort $w = w_1 \ldots w_m$ genau dann zu der Sprache $L_{i,j}^k$, wenn

$$\delta^*(i, w) = j \quad \text{und} \quad \forall s \in \{1, 2, \ldots, m-1\} : \delta^*(i, w_1 \ldots w_s) \leq k.$$

Werden beim Lesen eines Wortes w startend in Zustand 4 beispielsweise die Zustände $4, 3, 2, 4, 5$ durchlaufen, so gehört das Wort w zu der Sprache $L_{4,5}^4$, nicht aber zu der Sprache $L_{4,5}^3$. Als weiteres Beispiel betrachten wir den folgenden DFA.



Für diesen Automaten gilt beispielsweise

$$\begin{aligned}
L^1_{1,3} &= \emptyset,\\
L^0_{4,3} &= \{0\},\\
L^2_{1,3} &= \{01^n0 \mid n \in \mathbb{N}_0\},\\
L^4_{1,3} &= \{01^n0 \mid n \in \mathbb{N}_0\} \cup \{10\},\\
L^1_{2,3} &= \{0\},\\
L^2_{2,3} &= \{1^n0 \mid n \in \mathbb{N}_0\},\\
L^1_{2,2} &= \{\varepsilon\},\\
L^2_{2,2} &= \{1^n \mid n \in \mathbb{N}_0\}.
\end{aligned}$$

Wir werden für jede Sprache $L^k_{i,j}$ einen regulären Ausdruck $R^k_{i,j}$ entwerfen. Diese regulären Ausdrücke entsprechen den oben erwähnten Teilproblemen, die zu lösen sind. Zunächst überlegen wir uns, wie wir mithilfe dieser regulären Ausdrücke einen regulären Ausdruck für die Sprache L des DFA M entwerfen können. Dazu nutzen wir die Ausdrücke $L^n_{i,j}$. Da per Definition jeder Zustand $q \in Q$ die Bedingung $q \leq n$ erfüllt, gilt

$$w \in L^n_{i,j} \iff \delta^*(i,w) = j.$$

Ein Wort w wird von dem DFA M genau dann akzeptiert, wenn er sich startend im Startzustand $q_0 = 1$ nach dem Lesen von w in einem akzeptierenden Zustand befindet. Somit gilt

$$w \in L \iff \exists q \in F : \delta^*(1,w) = q \iff w \in \cup_{q\in F} L^n_{1,q}.$$

Da wir für jede Sprache $L^n_{1,q}$ einen regulären Ausdruck $R^n_{1,q}$ konstruieren werden, können wir auch einen regulären Ausdruck für die obige Vereinigung angeben. Gilt $F = \{f_1, \ldots, f_z\}$, so ist dies der reguläre Ausdruck

$$R^n_{1,f_1} + R^n_{1,f_2} + \ldots + R^n_{1,f_z}.$$

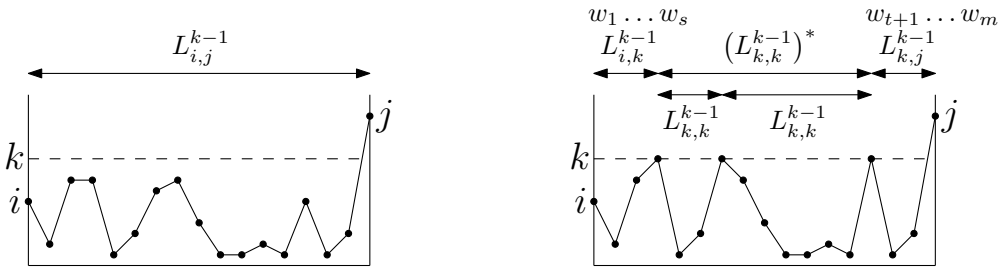
Haben wir also alle Teilprobleme gelöst und für jede Sprache $L^k_{i,j}$ einen regulären Ausdruck $R^k_{i,j}$ entworfen, so können wir auch einen regulären Ausdruck für die Sprache L angeben.

Wir werden zunächst die regulären Ausdrücke $R^0_{i,j}$ entwerfen. Seien $i, j \in Q$ beliebig. Beim Lesen von Wörtern aus der Sprache $L^0_{i,j}$ darf der Automat startend in Zustand i nur Zwischenzustände $q \leq 0$ durchlaufen. Solche Zustände gibt es per Definition aber nicht, weswegen $L^0_{i,j}$ genau die Wörter enthält, die ohne Zwischenzustand direkt von Zustand i in Zustand j führen. Dies sind ausschließlich Wörter der Länge eins. Es gilt

$$L^0_{i,j} = \{a \in \Sigma \mid \delta(i,a) = j\}.$$

Da Σ ein endliches Alphabet ist, enthält $L_{i,j}^0$ nur endlich viele Wörter. Sei $L_{i,j}^0 = \{a_1, \ldots, a_z\}$ mit $z \leq |\Sigma|$, dann wird die Sprache $L_{i,j}^0$ durch den regulären Ausdruck $a_1 + \ldots + a_z$ beschrieben.

Nun zeigen wir, wie man für $k \geq 1$ basierend auf den Ausdrücken $R_{i,j}^{k-1}$ die Ausdrücke $R_{i,j}^k$ konstruieren kann. Seien dazu $i, j \in Q$ und $k \geq 1$ beliebig und sei ein Wort $w = w_1 \ldots w_m \in L_{i,j}^k$ gegeben. Wir wissen, dass M startend in Zustand i beim Lesen von w den Zustand j erreicht und als Zwischenzustände nur Zustände $q \leq k$ besucht. Wir unterscheiden, ob für alle besuchten Zwischenzustände q sogar $q \leq k-1$ gilt oder ob der Zustand k mindestens einmal erreicht wird. Diese beiden Fälle sind in den folgenden Abbildungen dargestellt, in denen die Punkte die besuchten Zustände darstellen, die der DFA M startend in Zustand i beim Lesen des Wortes w besucht.



Besucht der DFA M nur Zwischenzustände $q \leq k-1$, so gehört das Wort w zu der Sprache $L_{i,j}^{k-1}$, für die wir bereits den regulären Ausdruck $R_{i,j}^{k-1}$ konstruiert haben. Ansonsten betrachten wir den ersten und letzten Besuch des Zustandes k. Es seien dafür $s, t \in \{1, \ldots, m-1\}$ so gewählt, dass sich der DFA M nach dem Lesen des Präfixes $w_1 \ldots w_s$ zum ersten Mal in Zustand k befindet und nach dem Lesen des Präfixes $w_1 \ldots w_t$ zum letzten Mal. Dann gehört das Wort $w_1 \ldots w_s$ zu der Sprache $L_{i,k}^{k-1}$ und das Wort $w_{t+1} \ldots w_m$ gehört zu der Sprache $L_{k,j}^{k-1}$. Das Wort $w_{s+1} \ldots w_t$ führt den Automaten vom Zustand k zurück in den Zustand k. Es gehört aber nicht unbedingt zu der Sprache $L_{k,k}^{k-1}$, da der Zustand k wie in der obigen rechten Abbildung als Zwischenzustand wieder auftreten darf. Damit können wir aber argumentieren, dass das Wort $w_{s+1} \ldots w_t$ zum Kleeneschen Abschluss $\left(L_{k,k}^{k-1}\right)^*$ gehören muss. Damit ist gezeigt, dass das Wort w zu der Sprache

$$L_{i,k}^{k-1} \cdot \left(L_{k,k}^{k-1}\right)^* \cdot L_{k,j}^{k-1}$$

gehört.

Betrachten wir beide Fälle, so folgt, dass die Sprache $L_{i,j}^k$ die Vereinigung der Sprachen $L_{i,j}^{k-1}$ und $L_{i,k}^{k-1} \cdot \left(L_{k,k}^{k-1}\right)^* \cdot L_{k,j}^{k-1}$ ist. Basierend auf den regulären Ausdrücken, die wir bereits konstruiert haben, können wir somit den folgenden regulären Ausdruck $R_{i,j}^k$ für die Sprache $L_{i,j}^k$ angeben:

$$R_{i,j}^k = R_{i,j}^{k-1} + R_{i,k}^{k-1}\left(R_{k,k}^{k-1}\right)^* R_{k,j}^{k-1}.$$

Damit ist der Beweis abgeschlossen, da wir auf diese Weise für jedes Tripel $i, j \in Q$ und $k \in \{0, 1, \ldots, n\}$ einen regulären Ausdruck für die Sprache $L_{i,j}^k$ konstruieren können. □

Das folgende Theorem folgt direkt aus den Lemmata 3.21 und 3.22.

Theorem 3.23. *Die Klasse der Sprachen, die mit regulären Ausdrücken beschrieben werden können, stimmt mit der Klasse der regulären Sprachen überein.*

Kapitel 4

Ausgewählte Themen der Mathematik

In diesem Abschnitt werden wir noch einige weitere Themen der Mathematik besprechen, die auf den Grundlagen aus Kapitel 2 aufbauen und für ein Studium der Informatik benötigt werden.

4.1 Abzählbare und überabzählbare Mengen

Wir haben in dieser Vorlesung bereits in einigen Beweisen die folgende Aussage benutzt, die als *Schubfachprinzip* (engl. *pigeonhole principle*) bezeichnet wird.

> Es sei $n \in \mathbb{N}$ mit $n \geq 2$ beliebig. Werden n Objekte auf höchstens $n-1$ Schubladen verteilt, so gibt es stets mindestens eine Schublade, die mehr als ein Objekt enthält.

Beispielsweise haben wir im Beweis des Pumping-Lemmas die Folge $q_0, q_1, \ldots, q_n$ von Zuständen betrachtet, die ein DFA beim Lesen eines Wortes der Länge n durchläuft. Wir haben ausgenutzt, dass sich für jeden DFA mit n Zuständen mindestens ein Zustand in dieser Folge wiederholen muss. In diesem Falle entsprechen die durchlaufenen Zustände $q_0, q_1, \ldots, q_n$ den $n+1$ zu verteilenden Objekten und die Schubladen sind die n Zustände des DFA.

Auch im Alltag können wir das Schubfachprinzip anwenden. Sind beispielsweise 13 Personen in einem Raum, so wissen wir, dass zwei von ihnen im selben Monat Geburtstag haben. Ebenso können wir schlussfolgern, dass es zwei Münchener geben muss, die exakt die gleiche Anzahl an Haaren auf dem Kopf haben. Ein Mensch hat typischerweise nicht mehr als 150.000 Haare und er hat ganz sicher weniger als eine Million. Nun hat München aber mehr als eine Million Einwohner, weshalb mindestens zwei davon dieselbe Anzahl an Haaren haben.

Formal können wir das Schubfachprinzip auch wie folgt ausdrücken.

Theorem 4.1 (Schubfachprinzip)**.** *Es seien A und B endliche Mengen mit $|A| > |B|$. Dann gibt es keine injektive Abbildung $f\colon A \to B$. Das heißt für jede Abbildung $f\colon A \to B$ gibt es Elemente $a \in A$ und $a' \in A$ mit $a \neq a'$ und $f(a) = f(a')$.*

Wir interessieren uns nun für die Frage, ob eine vergleichbare Aussage auch für unendliche Mengen möglich ist. Dazu holen wir ein wenig aus und erinnern uns zunächst an Definition 2.14, die besagt, dass zwei Mengen gleichmächtig sind, wenn es eine bijektive Abbildung zwischen ihnen gibt. Basierend auf dieser Definition wollen wir die Kardinalität von verschiedenen unendlichen Mengen miteinander vergleichen.

Definition 4.2. *Eine Menge M heißt* abzählbar unendlich*, wenn sie dieselbe Kardinalität wie die Menge $\mathbb{N}$ der natürlichen Zahlen besitzt, d. h. wenn es eine bijektive Abbildung $f\colon \mathbb{N} \to M$ gibt. Eine Menge heißt* abzählbar*, wenn sie entweder endlich oder abzählbar unendlich ist. Eine unendliche Menge, die nicht abzählbar unendlich ist, heißt* überabzählbar.

Intuitiv ist eine Menge genau dann abzählbar, wenn man ihre Elemente mit den natürlichen Zahlen durchnummerieren kann. Wir definieren eine Relation $\sim$ auf der Menge aller Mengen. Dabei gelte $A \sim B$ genau dann, wenn A und B gleichmächtig sind, d. h. wenn es eine bijektive Abbildung zwischen A und B gibt. Der Leser sollte sich als Übung überlegen, dass es sich bei dieser Relation um eine Äquivalenzrelation handelt. Wegen der Transitivität von $\sim$ ist eine Menge B abzählbar unendlich, wenn es eine bijektive Abbildung zwischen B und einer abzählbar unendlichen Menge A gibt.

Zunächst weisen wir einige nützliche Eigenschaften abzählbarer Mengen nach.

Theorem 4.3. *Es sei B eine abzählbar unendliche Menge und $A \subseteq B$. Dann ist A endlich oder ebenfalls abzählbar unendlich.*

Beweis. Ist A endlich, so ist die Aussage offensichtlich wahr. Uns interessiert also nur der Fall, dass A eine unendliche Menge ist. Da die Menge B abzählbar ist, gibt es eine bijektive Abbildung $f\colon \mathbb{N} \to B$. Um das Theorem zu beweisen, geben wir eine bijektive Abbildung $g\colon \mathbb{N} \to A$ an. Dazu definieren wir zunächst eine Hilfsfunktion $h\colon \mathbb{N} \to \mathbb{N}$. Es sei $h(1)$ die kleinste Zahl $m \in \mathbb{N}$, für die $f(m) \in A$ gilt, also

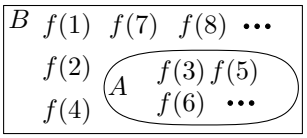
$$h(1) = \min\{m \in \mathbb{N} \mid f(m) \in A\}.$$

Außerdem sei $h(n)$ für $n \geq 2$ die kleinste Zahl $m \in \mathbb{N}$ mit $m > h(n-1)$, für die $f(m) \in A$ gilt, also

$$h(n) = \min\{m \in \mathbb{N} \mid m > h(n-1) \text{ und } f(m) \in A\}.$$

Da die Menge A unendlich ist, ist die Funktion h wohldefiniert.

Diese Definition wird in der folgenden Abbildung illustriert. Für das dort gezeigte Beispiel gilt $h(1) = 3$, $h(2) = 5$ und $h(3) = 6$.



Mithilfe der Funktion h können wir nun die bijektive Abbildung $g\colon \mathbb{N} \to A$ angeben. Wir definieren dazu $g(n) = f(h(n))$ für alle $n \in \mathbb{N}$. Den formalen Beweis, dass diese Funktion bijektiv ist, überlassen wir dem Leser als Übung. Intuitiv sollte man sich dafür klar machen, dass die Funktion g die Elemente aus A durchnummeriert. Dabei kommt ein Element $a \in A$ in der Nummerierung g genau dann vor einem Element $a' \in A$, wenn a in der Nummerierung f vor dem Element a' kommt. □

Theorem 4.4. *Die Mengen $\mathbb{Z}$ und $\mathbb{N} \times \mathbb{N}$ sind abzählbar unendlich.*

Beweis. Wir betrachten die Funktion $f\colon \mathbb{N} \to \mathbb{Z}$ mit

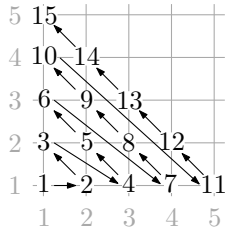
$$f(n) = \begin{cases} \frac{n}{2} & \text{falls } n \text{ gerade,} \\ -\frac{n-1}{2} & \text{falls } n \text{ ungerade.} \end{cases}$$

Die ersten Werte dieser Funktion sind $0, 1, -1, 2, -2, 3, -3, \ldots$. Es ist eine leichte Übung zu zeigen, dass die Funktion f bijektiv ist.

Zum Beweis, dass auch $\mathbb{N} \times \mathbb{N}$ abzählbar unendlich ist, nutzen wir die *Cantorsche Paarungsfunktion* $g\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Diese ist durch die Abbildungsvorschrift

$$g(x, y) = \frac{(x + y - 2)(x + y - 1)}{2} + y$$

definiert. Das folgende Bild illustriert diese Abbildung.



Wir überlassen den formalen Nachweis, dass die Cantorsche Paarungsfunktion bijektiv ist, dem Leser. Gemäß Definition 4.2 müssen wir eigentlich eine bijektive Abbildung von $\mathbb{N}$ nach $\mathbb{N} \times \mathbb{N}$ angeben und nicht umgekehrt. Dies spielt aber keine Rolle, da die Umkehrabbildung einer bijektiven Abbildung ebenfalls bijektiv ist. □

Wir überlassen den Beweis des folgenden Theorems dem Leser als Übung. Als Tipp sei erwähnt, dass das Theorem mit sehr ähnlichen Methoden wie das vorangegangene Theorem 4.4 bewiesen werden kann.

Theorem 4.5. *Es seien A und B zwei abzählbar unendliche Mengen. Dann sind auch die Mengen $A \cup B$ und $A \times B$ abzählbar unendlich.*

Möchte man zeigen, dass eine Menge B abzählbar unendlich ist, so muss man eine bijektive Abbildung zwischen einer abzählbar unendlichen Menge A und der Menge B angeben. In vielen Fällen ist es deutlich einfacher, eine surjektive Abbildung $f\colon A \to B$ zu finden. Das bedeutet, dass die Elemente von B durchnummeriert werden, dass ein Element aus B aber mehrere Nummern erhalten kann. Die Intuition sagt uns, dass B dann endlich oder abzählbar unendlich sein muss.

Theorem 4.6. *Es sei A eine abzählbar unendliche Menge und es sei $f\colon A \to B$ eine surjektive Abbildung. Dann ist die Menge B endlich oder abzählbar unendlich.*

Beweis. Ist B endlich, so ist die Aussage des Theorems erfüllt. Wir gehen also davon aus, dass B unendlich ist. Da A abzählbar unendlich ist, existiert eine bijektive Abbildung $g\colon \mathbb{N} \to A$. Darauf aufbauend definieren wir eine Abbildung $h\colon \mathbb{N} \to B$ durch $h(n) = f(g(n))$. Da f surjektiv und g bijektiv ist, ist die Abbildung h ebenfalls surjektiv.

Im Allgemeinen ist die Abbildung h nicht injektiv, wir können sie aber injektiv machen, indem wir bereits aufgezählte Elemente überspringen. Formal definieren wir eine Funktion $\pi\colon \mathbb{N} \to \mathbb{N}$ mit $\pi(1) = 1$ und

$$\pi(n) = \min\{k \in \mathbb{N} \mid h(k) \in B \setminus \{h(\pi(1)), \ldots, h(\pi(n-1))\}\}$$

für $n \geq 2$. Da die Menge B unendlich ist, ist die Funktion π wohldefiniert. Wir definieren basierend auf π die Funktion $h'\colon \mathbb{N} \to B$ mit $h'(n) = h(\pi(n))$. Der Leser sollte sich überlegen, dass die Funktion $h'\colon \mathbb{N} \to B$ bijektiv ist. Damit ist gezeigt, dass die Menge B abzählbar unendlich ist. □

Theorem 4.7. *Die Menge $\mathbb{Q}$ der rationalen Zahlen ist abzählbar unendlich.*

Beweis. Gemäß der Theoreme 4.4 und 4.5 ist die Menge $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$ abzählbar unendlich. Es ist leicht zu sehen, dass die Abbildung $f\colon \mathbb{Z}\times(\mathbb{Z}\setminus\{0\}) \to \mathbb{Q}$ mit $f(a,b) = \frac{a}{b}$ surjektiv ist. Aus Theorem 4.6 folgt damit, dass die Menge $\mathbb{Q}$ abzählbar unendlich ist. □

Nach den vorangegangenen Beispielen könnte man die Hypothese aufstellen, dass alle unendlichen Mengen abzählbar unendlich sind. Das folgende Theorem zeigt, dass dies nicht der Fall ist.

Theorem 4.8. *Die Menge $\mathbb{R}$ der reellen Zahlen ist überabzählbar.*

Beweis. Wir weisen das Theorem mit einem Widerspruchsbeweis nach und wenden eine sehr nützliche Technik an, die *Diagonalisierung* genannt wird. Nehmen wir an, die reellen Zahlen seien abzählbar unendlich. Dann folgt aus Theorem 4.3, dass insbesondere die Menge $(0,1) = \{x \in \mathbb{R} \mid 0 < x < 1\}$ abzählbar unendlich ist. Es gibt also eine bijektive Abbildung $f\colon \mathbb{N} \to (0,1)$, die alle reellen Zahlen zwischen 0 und 1 durchnummeriert. Für $n \in \mathbb{N}$ und $i \in \mathbb{N}$ sei $a_n = f(n)$ und a_{ni} bezeichne die i-te Dezimalstelle von a_n. Schreiben wir alle reellen Zahlen in der durch f vorgegebenen Reihenfolge untereinander, so erhalten wir das folgende Bild.

$$\begin{aligned} a_1 &= 0, a_{11}a_{12}a_{13}\ldots \\ a_2 &= 0, a_{21}a_{22}a_{23}\ldots \\ a_3 &= 0, a_{31}a_{32}a_{33}\ldots \\ &\vdots \end{aligned}$$

Wir definieren nun eine reelle Zahl $b \in (0,1)$. Für $i \in \mathbb{N}$ bezeichnen wir mit b_i die i-te Dezimalstelle von b. Wir setzen

$$b_i = \begin{cases} 1 & \text{falls } a_{ii} \neq 1, \\ 2 & \text{falls } a_{ii} = 1. \end{cases}$$

Da die Zahl b in dem Intervall $(0,1)$ liegen soll, steht vor dem Komma eine Null.

Die Zahl b gehört zu dem Intervall $(0,1)$. Damit muss es wegen der Bijektivität von f ein $n \in \mathbb{N}$ mit $a_n = f(n) = b$ geben. Dies kann aufgrund der Definition von b aber nicht sein, denn es gilt $b_n \neq a_{nn}$. Das heißt, für jedes $n \in \mathbb{N}$ gilt $b \neq a_n$, da sich b und a_n in der n-ten Dezimalstelle unterscheiden. Damit erhalten wir einen Widerspruch dazu, dass f bijektiv ist. □

Wir können nun das folgende Schubfachprinzip für unendliche Mengen formulieren.

Theorem 4.9. *Es sei A eine überabzählbare Menge und es sei B eine abzählbar unendliche Menge. Dann gibt es keine injektive Abbildung $f\colon A \to B$.*

Beweis. Wir führen einen Widerspruchsbeweis und gehen davon aus, dass eine injektive Abbildung $f\colon A \to B$ existiert. Die Abbildung f ist nicht notwendigerweise surjektiv, aber wir können sie wie in Abschnitt 2.4.2 beschrieben dadurch surjektiv machen, dass wir die Zielmenge auf die Bildmenge

$$f(A) = \{f(a) \mid a \in A\} \subseteq B$$

einschränken. Die Funktion $g\colon A \to f(A)$ mit $g(a) = f(a)$ für alle $a \in A$ ist demnach bijektiv. Da die Bildmenge $f(A)$ eine Teilmenge der abzählbar unendlichen Menge B ist, ist sie gemäß Theorem 4.3 endlich oder abzählbar unendlich. Da es eine bijektive Abbildung zwischen den Mengen $f(A)$ und A gibt, bedeutet das, dass auch die Menge A endlich oder abzählbar unendlich sein muss. Das ist ein Widerspruch zu der Annahme, dass A überabzählbar ist. □

Eine überaus interessante Frage für die Informatik ist es, ob es für jede Funktion $f\colon \mathbb{N} \to \mathbb{N}$ ein Programm (z. B. in Java) gibt, das die Funktion f berechnet.

Theorem 4.10. *Es gibt eine Funktion $f\colon \mathbb{N} \to \mathbb{N}$, die von keinem Programm berechnet werden kann.*

Beweis. Es sei F die Menge aller Funktionen $f\colon \mathbb{N} \to \mathbb{N}$ und es sei P die Menge aller Programme in unserer Lieblingsprogrammiersprache. Ein Programm ist eine endliche Folge von Zeichen über einem endlichen Alphabet Σ. Es gilt somit $P \subseteq \Sigma^*$.

Wir können wieder mithilfe von Diagonalisierung nachweisen, dass die Menge F überabzählbar ist. Nehmen wir an, dass F abzählbar ist. Dann können wir alle Funktionen aus F durchnummerieren. Sei $f_1, f_2, f_3, \ldots$ diese Nummerierung. Wir definieren nun eine Funktion $g\colon \mathbb{N} \to \mathbb{N}$ mit

$$g(n) = \begin{cases} 1 & \text{falls } f_n(n) \neq 1, \\ 2 & \text{falls } f_n(n) = 1. \end{cases}$$

Dann gilt $g(n) \neq f_n(n)$ für alle $n \in \mathbb{N}$. Das bedeutet, die Funktion g kommt in der Aufzählung $f_1, f_2, f_3, \ldots$ nicht vor. Dies ist ein Widerspruch, da g selbst eine Funktion aus der Menge F ist.

Die Menge $P \subseteq \Sigma^*$ ist abzählbar unendlich. Wegen Theorem 4.3 genügt es zu zeigen, dass Σ^* abzählbar unendlich ist. Dazu geben wir eine bijektive Abbildung zwischen $\mathbb{N}$ und Σ^* an. Um diese Abbildung zu beschreiben, genügt es eine Nummerierung aller Wörter aus Σ^* anzugeben. Wir nummerieren die Wörter dabei gemäß ihrer Länge und Wörter derselben Länge alphabetisch. Für $\Sigma = \{0, 1\}$ zählen wir die Wörter aus Σ^* wie folgt auf: $\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots$

Es sei nun $S\colon P \to F$ die Abbildung, die jedem Programm $p \in P$ die Funktion $S(p) \in F$ zuordnet, die p berechnet. Theorem 4.6 besagt, dass die Funktion S nicht surjektiv ist, da sonst im Widerspruch zu obiger Argumentation folgen würde, dass die Menge F aller Funktionen abzählbar unendlich ist. Das bedeutet, es gibt eine Funktion $f \in F$, für die es kein Programm $p \in P$ mit $S(p) = f$ gibt. Die Funktion f wird also von keinem Programm berechnet. □

Der vorangegangene Beweis von Theorem 4.10 ist nicht konstruktiv. Das bedeutet, dass zwar die Existenz einer nicht berechenbaren Funktion nachgewiesen wird, dass aber keine solche Funktion explizit konstruiert wird. Dies lässt noch die Möglichkeit offen, dass zumindest alle interessanten Funktionen, die in der Praxis von Bedeutung sind, berechnet werden können. Leider ist dies aber nicht der Fall. Wir werden in der Vorlesung *Algorithmen und Berechnungskomplexität II* für viele konkrete Probleme nachweisen, dass sie von keinem Programm gelöst werden können. Dazu zählt insbesondere das in der Einleitung bereits angesprochene Halteproblem.

4.2 Abzählende Kombinatorik

In diesem Abschnitt geben wir eine kurze Einführung in das Gebiet der *abzählenden Kombinatorik*. Dabei geht es darum, die Anzahl von möglichen Anordnungen und Auswahlen verschiedener Objekte zu zählen. Die folgenden Fragestellungen sind typische Beispiele.

- Wie viele verschiedene mögliche Ausgänge gibt es beim Lotto (6 aus 49)?
- Wie viele verschiedene Wege von der Ecke links oben zu der Ecke rechts unten gibt es in einem $(n \times m)$-Gitter?
- Wie viele verschiedene Reihenfolgen gibt es, in denen eine gegebene Menge von n Städten besucht werden kann?
- Wie viele verschiedene Äquivalenzrelationen gibt es auf einer Menge mit n Elementen?

Diese und ähnliche Fragen spielen bei der Analyse von Algorithmen oft eine große Rolle. Deswegen ist es wichtig, zumindest ein grundlegendes Verständnis dafür zu entwickeln.

In diesem Abschnitt treten oft *Fakultäten* und *Binomialkoeffizienten* auf.

Definition 4.11. *Für $n \in \mathbb{N}_0$ sei $n!$ (gesprochen: „n Fakultät“) definiert als*

$$n! = \prod_{k=1}^{n} k.$$

Gemäß der Konvention aus Abschnitt 2.2.4 gilt $0! = 1$.

Für $n \in \mathbb{N}_0$ und $k \in \{0, \ldots, n\}$ sei der Binomialkoeffizient $\binom{n}{k}$ (gesprochen: „n über k“) definiert als

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}.$$

Für $k > n$ und $k < 0$ gelte außerdem $\binom{n}{k} = 0$.

Es gilt beispielsweise

$$\binom{5}{3} = \frac{5!}{2! \cdot 3!} = \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{(2 \cdot 1) \cdot (3 \cdot 2 \cdot 1)} = \frac{5 \cdot 4}{2} = 10.$$

Mithilfe von Binomialkoeffizienten kann man den *binomischen Lehrsatz* formulieren.

Theorem 4.12. *Für alle $n \in \mathbb{N}_0$ und $x, y \in \mathbb{R}$ gilt*

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Den Beweis dieses Satzes überlassen wir dem Leser als Übung. Dies ist eine gute Gelegenheit, noch einmal das Prinzip der vollständigen Induktion zu wiederholen.

Die folgenden beiden Formeln für Binomialkoeffizienten sind oft sehr nützlich.

Theorem 4.13. *Für alle $n, k \in \mathbb{N}_0$ mit $k \leq n$ gilt*

$$\binom{n}{k} = \binom{n}{n-k}$$

und

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}.$$

Beweis. Der Beweis folgt durch Umformen der Terme. Es gilt

$$\binom{n}{n-k} = \frac{n!}{(n-k)! \cdot (n-(n-k))!} = \frac{n!}{(n-k)! \cdot k!} = \binom{n}{k}$$

und für $k \geq 1$ gilt

$$\binom{n}{k} + \binom{n}{k-1} = \frac{n!}{k! \cdot (n-k)!} + \frac{n!}{(k-1)! \cdot (n-k+1)!}$$

$$= \frac{n! \cdot (n-k+1) + n! \cdot k}{k! \cdot (n-k+1)!}$$
$$= \frac{(n+1)!}{k! \cdot (n-k+1)!} = \binom{n+1}{k}.$$

Für $k = 0$ folgt die zweite Gleichung aus $\binom{n+1}{0} = \binom{n}{0} = 1$ und $\binom{n}{-1} = 0$. □

Ganz grundlegend ist die Frage, wie viele mögliche Reihenfolgen es gibt, in denen n Objekte angeordnet werden können. Für die Menge $M = \{a, b, c\}$ gibt es beispielsweise die 6 Anordnungen (a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b) und (c, b, a). Jede solche Anordnung nennen wir eine *Permutation* der Menge M und allgemein besitzt eine Menge M mit n Elementen genau $n!$ Permutationen. Dies liegt daran, dass es für das erste Element der Permutation n mögliche Wahlen gibt. Steht das erste Element fest, so gibt es für das zweite Element nur noch $n-1$ mögliche Wahlen. Stehen die ersten beiden Elemente fest, so gibt es für das dritte Element nur noch $n-2$ mögliche Wahlen und so weiter. Multipliziert man diese Wahlmöglichkeiten, so erhält man $n \cdot (n-1) \cdot \ldots \cdot 1 = n!$ mögliche Permutationen.

Oft ist man an der Frage interessiert, wie viele Teilmengen einer bestimmten Kardinalität k eine Menge mit n Elementen besitzt. Wir nennen eine solche Teilmenge auch eine *k-Teilmenge*.

Theorem 4.14. *Die Anzahl verschiedener k-Teilmengen einer Menge mit n Elementen beträgt $\binom{n}{k}$.*

Beweis. Wir bezeichnen mit $a(n, k)$ die gesuchte Anzahl. Für alle $k > 0$ ist $a(0, k) = 0$, da es keine Teilmenge der leeren Menge mit $k > 0$ Elementen gibt. Außerdem ist $a(n, 0) = 1$ für alle $n \in \mathbb{N}_0$, da es für jede Menge genau eine Teilmenge der Kardinalität 0 gibt, nämlich die leere Menge.

Für $n \geq 1$ und $k \geq 1$ benutzen wir eine ähnliche Argumentation wie im Beweis von Theorem 2.8. Wir gehen davon aus, dass eine beliebige Menge M mit n Elementen gegeben ist, und wir bezeichnen mit $x \in M$ ein beliebiges Element aus M. Sei $N \subseteq M$ eine k-Teilmenge von M. Dann können zwei Fälle eintreten.

1. Gilt $x \notin N$, so ist N eine k-Teilmenge von $M \setminus \{x\}$.
   Die Menge der k-Teilmengen N von M mit $x \notin N$ entspricht demnach der Menge der k-Teilmengen von $M \setminus \{x\}$.
2. Gilt $x \in N$, so ist $N \setminus \{x\}$ eine $(k-1)$-Teilmenge von $M \setminus \{x\}$.
   Die Menge der k-Teilmengen N von M mit $x \in N$ kann durch $N \mapsto N \setminus \{x\}$ bijektiv auf die Menge der $(k-1)$-Teilmengen von $M \setminus \{x\}$ abgebildet werden.

Diese Überlegung beweist, dass für $n \geq 1$ und $k \geq 1$ die Rekursionsformel $a(n, k) = a(n-1, k) + a(n-1, k-1)$ gilt.

Wir stellen fest, dass die Zahlen $a(n, k)$ denselben Anfangsbedingungen für $n = 0$ oder $k = 0$ und derselben Rekursionsformel für $n \geq 1$ und $k \geq 1$ genügen wie die Binomialkoeffizienten $\binom{n}{k}$ (vergleiche Theorem 4.13). Man überlegt sich leicht, dass die

Zahlen $a(n,k)$ und die Binomialkoeffizienten $\binom{n}{k}$ durch diese Anfangsbedingungen und die Rekursionsformel eindeutig bestimmt werden. Dies bedeutet, dass $a(n,k) = \binom{n}{k}$ für jedes $n \in \mathbb{N}_0$ und jedes $k \in \{0, 1, \ldots, n\}$ gilt. □

Viele Fragen, die uns in der abzählenden Kombinatorik interessieren, können in abstrakten Urnenmodellen formuliert werden. In einem solchen Modell ist eine Urne gegeben, in der sich n Kugeln befinden, die mit den Zahlen 1 bis n durchnummeriert sind. Nun wird k mal jeweils eine Kugel aus der Urne gezogen wird. Wir sind daran interessiert, wie viele verschiedene Ziehungen dabei auftreten können. Dabei unterscheiden wir vier Szenarien, je nachdem ob die gezogenen Kugeln direkt wieder zurück in die Urne gelegt werden und ob die Reihenfolge der gezogenen Kugeln von Bedeutung ist.

- *Ziehung ohne Zurücklegen mit Beachtung der Reihenfolge*
  Einmal gezogene Kugeln werden nicht zurück in die Urne gelegt, aber (anders als beim Lotto) ist die Reihenfolge der gezogenen Kugeln von Bedeutung. Beispielsweise werden die Ziehungen $(1,2)$ und $(2,1)$ als verschieden betrachtet. Eine Ziehung von k Kugeln wird in diesem Szenario auch *k-Permutation* genannt.

  Die Anzahl der verschiedenen Ziehungen beträgt in diesem Szenario $\frac{n!}{(n-k)!}$. Um dies einzusehen, überlegt man sich zunächst, dass es für die erste gezogene Kugel n Möglichkeiten gibt. Danach sind nur noch $n-1$ Kugeln in der Urne, weshalb es für die zweite gezogene Kugel nur noch $n-1$ Möglichkeiten gibt. Allgemein gibt es für die i-te Kugel $n-i+1$ Möglichkeiten, wenn die ersten $i-1$ Kugeln bereits feststehen. Multiplizieren wir die Möglichkeiten für die k gezogenen Kugeln, so erhalten wir

  $$n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1) = \frac{n!}{(n-k)!} = \binom{n}{k} \cdot k!.$$

  Für $k = n$ ergeben sich $n!$ verschiedene Ziehungen. Dies ist nicht überraschend, denn für $k = n$ werden alle Kugeln aus der Urne gezogen, wodurch eine Permutation auf den n Kugeln definiert wird.
- *Ziehung mit Zurücklegen mit Beachtung der Reihenfolge*
  Die gezogenen Kugeln werden direkt wieder zurück in die Urne gelegt und können später erneut gezogen werden. Außerdem ist wieder die Reihenfolge der gezogenen Kugeln von Bedeutung.

  Die Anzahl verschiedener Ziehungen beträgt in diesem Szenario n^k. Dies folgt mit derselben Argumentation wie oben. Der einzige Unterschied ist, dass es nun für jede Kugel n Möglichkeiten gibt, da die bereits gezogenen Kugeln erneut gezogen werden können. Formal kann man eine bijektive Abbildung zwischen der Menge der möglichen Ziehungen und der Menge $\{1, 2, \ldots, n\}^k$, deren Kardinalität n^k ist, angeben.
- *Ziehung ohne Zurücklegen ohne Beachtung der Reihenfolge*
  Einmal gezogene Kugeln werden nicht zurückgelegt und die Reihenfolge der gezogenen Kugeln wird nicht beachtet. Dieses Szenario entspricht dem Lottospiel „6 aus 49“.

Durch eine Ziehung wird in diesem Szenario eine k-Teilmenge von $\{1, \ldots, n\}$ bestimmt. Theorem 4.14 besagt, dass es $\binom{n}{k}$ solcher k-Teilmengen gibt.

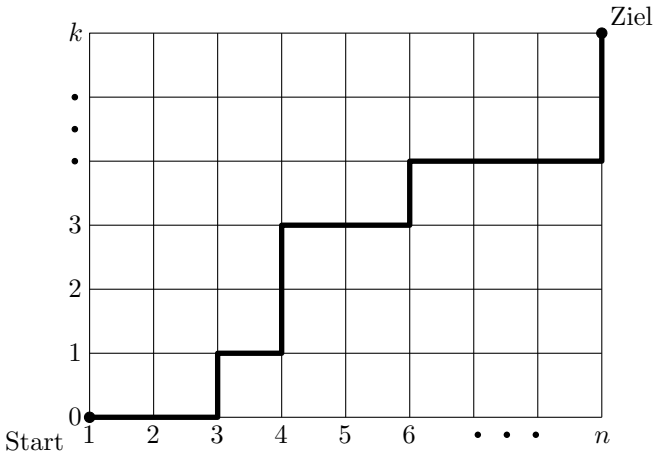
Wir stellen fest, dass die Anzahl von k-Permutationen um genau den Faktor $k!$ größer ist als die Anzahl von k-Teilmengen. Dies ergibt Sinn, denn wählt man eine k-Teilmenge aus, so gibt es $k!$ Möglichkeiten für die Reihenfolge, in der die Elemente gezogen werden.

- *Ziehung mit Zurücklegen ohne Beachtung der Reihenfolge*
  Die gezogenen Kugeln werden direkt wieder zurück in die Urne gelegt und können später erneut gezogen werden. Die Reihenfolge der gezogenen Kugeln ist aber nicht von Bedeutung.

  Eine Ziehung in diesem Szenario können wir als einen Vektor aus der Menge

$$M = \left\{ (x_1, x_2, \ldots, x_n) \in \mathbb{N}_0^n \;\middle|\; \sum_{i=1}^{n} x_i = k \right\}$$

  darstellen. In diesem Vektor gibt die Komponente x_i an, wie oft Kugel i gezogen wurde. Da die Reihenfolge nicht beachtet wird, gibt es eine bijektive Abbildung zwischen der Menge der Ziehungen und der Menge M. Es genügt also, die Kardinalität von M zu bestimmen. Um die Vektoren aus M anschaulich darzustellen, betrachten wir das Gitter $\{1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, k\}$. In diesem Gitter betrachten wir Wege vom Start $(1, 0)$ zum Ziel (n, k), die nur aus Schritten nach rechts oder nach oben bestehen. Ein solcher Weg ist in der folgenden Abbildung dargestellt.



  Wir können jeden Weg vom Start zum Ziel als einen Vektor $z \in M$ interpretieren: Da jeder solche Weg genau k vertikale Schritte von unten nach oben besitzt, können wir z_i als die Anzahl der vertikalen Schritte definieren, die an der x-Koordinate i erfolgen. Der Weg im obigen Beispiel codiert beispielsweise den Vektor $(0, 0, 1, 2, 0, 1, 0, 0, 2)$. Es ist leicht zu sehen, dass wir auf diese Weise eine bijektive Abbildung zwischen der Menge aller Wege vom Start zum Ziel und der Menge M erhalten.

  Wir müssen nun nur noch die Frage beantworten, wie viele Wege es gibt. Dazu beobachten wir zunächst, dass jeder Weg vom Start zum Ziel aus genau $n+k-1$

Schritten besteht, nämlich aus k Schritten nach oben und $n-1$ Schritten nach rechts. Die Schritte nach oben und die Schritte nach rechts können in einer beliebigen Reihenfolge erfolgen. Das bedeutet, es gibt genau so viele Wege wie es Möglichkeiten gibt, die k Schritte nach oben auf die insgesamt $n+k-1$ Schritte zu verteilen. Wir sind also an der Anzahl an Möglichkeiten interessiert von $n+k-1$ Schritten k Schritte auszuwählen, d. h. an der Anzahl von k-Teilmengen einer Menge mit $n+k-1$ Elementen. Gemäß Theorem 4.14 gibt es $\binom{n+k-1}{k}$ solcher k-Teilmengen. Dies entspricht der Anzahl der verschiedenen Ziehungen in diesem Szenario.

Wir fassen die vier betrachteten Szenarien noch einmal in der folgenden Tabelle zusammen.

	ohne Reihenfolge	mit Reihenfolge
ohne Zurücklegen	$\binom{n}{k}$	$\frac{n!}{(n-k)!}$
mit Zurücklegen	$\binom{n+k-1}{k}$	n^k

Als nächstes betrachten wir einige Beispiele, in denen die obigen Formeln Anwendung finden.

- Wir werfen gleichzeitig fünf nicht unterscheidbare Würfel. Wie viele verschiedene Ergebnisse können dabei auftreten? Es handelt sich um das Urnenmodell mit Zurücklegen, aber ohne Beachtung der Reihenfolge. Dabei ist $n=6$ und $k=5$, da fünfmal eine Zahl aus der Menge $\{1,2,3,4,5,6\}$ gezogen wird. Die Anzahl verschiedener Ergebnisse ist also

$$\binom{6+5-1}{5} = \binom{10}{5} = \frac{10!}{5!\cdot 5!} = \frac{10\cdot 9\cdot 8\cdot 7\cdot 6}{5\cdot 4\cdot 3\cdot 2\cdot 1} = 252.$$

- Wir betrachten das Lottospiel „6 aus 49“. Dabei werden 6 Kugeln ohne Zurücklegen und ohne Beachtung der Reihenfolge aus einer Menge von 49 Kugeln gezogen. Es gibt also insgesamt

$$\binom{49}{6} = 13.983.816$$

mögliche Ziehungen. Da beim Lotto jede Ziehung dieselbe Wahrscheinlichkeit besitzt, beträgt die Wahrscheinlichkeit, 6 Richtige zu haben, nur $\frac{1}{13.983.816}$.

Wie wahrscheinlich ist es, genau 4 Richtige zu haben? Dazu zählen wir, wie viele Ziehungen es gibt, die mit einem gegebenen Tipp in genau 4 Zahlen übereinstimmen. Für den gegebenen Tipp wählen wir zunächst aus, welche vier Zahlen in der Ziehung vorkommen sollen. Dafür gibt es $\binom{6}{4}$ Möglichkeiten. Die Ziehung muss dann abgesehen von diesen vier ausgewählten Zahlen noch zwei weitere

Zahlen enthalten, die nicht im Tipp vorkommen. Es gibt 43 Zahlen, die nicht im Tipp enthalten sind, und somit gibt es $\binom{43}{2}$ Möglichkeiten, diese zwei Zahlen zu wählen. Die gesuchte Zahl an Ziehungen entspricht dem Produkt dieser Binomialkoeffizienten, da jede Wahl des einen mit jeder Wahl des anderen kombiniert werden kann. Es gibt also

$$\binom{6}{4} \cdot \binom{43}{2} = \frac{6 \cdot 5}{2} \cdot \frac{43 \cdot 42}{2} = 13.545$$

Ziehungen, die mit einem gegeben Tipp in genau 4 Zahlen übereinstimmen. Da es insgesamt 13.983.816 Ziehungen gibt, beträgt die Wahrscheinlichkeit, genau 4 Richtige zu haben, somit

$$\frac{13.545}{13.983.816} \approx 0{,}0009686 = 0{,}09686\%.$$

- Wie wahrscheinlich ist es, dass es in einer Gruppe von 23 Personen mindestens zwei Personen gibt, die am selben Tag des Jahres Geburtstag haben? Wir ignorieren der Einfachheit halber Schaltjahre und gehen davon aus, dass jede Person unabhängig von den anderen an einem uniform zufälligen[1] Tag des Jahres Geburtstag hat (es befinden sich also insbesondere keine Zwillinge unter den Personen).

  Wir gehen davon aus, dass die Personen durchnummeriert sind. Dann können wir die Geburtstage als einen Vektor aus der Menge $M = \{1, 2, \ldots, 365\}^{23}$ darstellen. Diese Menge enthält 365^{23} Elemente. Wie viele der Vektoren aus M haben die Eigenschaft, dass alle Einträge verschieden sind? Der Leser überlege sich, dass es genauso viele solche Vektoren gibt, wie es 23-Permutationen der Menge $\{1, 2, \ldots, 365\}$ gibt, also gemäß unserer obigen Überlegungen $\frac{365!}{342!}$.

  Sind alle Geburtstage uniform zufällig und unabhängig gewählt, so wird durch die Geburtstage ein uniform zufälliges Element der Menge $M = \{1, 2, \ldots, 365\}^{23}$ bestimmt. Der Anteil von Vektoren aus M mit paarweise verschiedenen Einträgen beträgt gemäß der obigen Überlegungen

  $$\frac{\frac{365!}{342!}}{365^{23}} \approx 0{,}493 = 49{,}3\%.$$

  Das bedeutet, die Wahrscheinlichkeit, dass 23 Personen an paarweise verschiedenen Tagen Geburtstag haben, ist kleiner als 50%. Diese Erkenntnis wird auch als *Geburtstagsparadoxon* bezeichnet. Dabei handelt es sich nicht um ein Paradoxon im eigentlichen Sinne, viele Menschen sind aber verblüfft, dass bereits 23 Personen genügen, um mit einer Wahrscheinlichkeit von mehr als 50% einen doppelten Geburtstag zu haben.

  Warum sind wir oben davon ausgegangen, dass die Personen durchnummeriert sind? Auf den ersten Blick scheint es so, als hätten wir für die Argumentation

[1] „Uniform zufällig“ bedeutet, dass jeder Tag dieselbe Wahrscheinlichkeit $\frac{1}{365}$ besitzt, ausgewählt zu werden.

auch genauso gut den Fall betrachten können, dass die Personen nicht unterscheidbar sind. Dann wäre die Zahl der möglichen Konfigurationen nicht 365^{23}, sondern $\binom{365+23-1}{23}$, weil wir das Urnenmodell mit Zurücklegen und ohne Beachtung der Reihenfolge zugrunde legen müssen. Für die Zahl der möglichen Konfigurationen ohne doppelte Geburtstage legen wir in diesem Fall das Urnenmodell ohne Zurücklegen und ohne Beachtung der Reihenfolge zugrunde. Dementsprechend gibt es $\binom{365}{23}$ solche Konfigurationen. Nun liegt die Vermutung nahe, dass die Wahrscheinlichkeit, keinen doppelten Geburtstag zu sehen,

$$\frac{\binom{365}{23}}{\binom{365+23-1}{23}}$$

beträgt. Dies stimmt jedoch nicht, da nicht mehr jede Konfiguration dieselbe Wahrscheinlichkeit besitzt. Es wird also nicht mehr eine Konfiguration uniform zufällig aus der Menge aller Konfigurationen ausgewählt. In diesem Falle entspricht der Quotient nicht mehr der Wahrscheinlichkeit. Dies kann man sich am besten anhand eines Beispiels überlegen. Haben wir nur zwei Personen, so beträgt die Wahrscheinlichkeit, dass beide am 1. Januar Geburtstag haben, $1/365^2$. Die Wahrscheinlichkeit, dass einer von beiden am 1. Januar und der andere am 2. Januar Geburtstag hat, beträgt aber $2/365^2$, da es zwei Wahlmöglichkeiten für dieses Ereignis gibt (Person 1 hat am 1. Januar Geburtstag und Person 2 am 2. Januar oder umgekehrt).

4.3 Algebraische Strukturen

Eine *Verknüpfung* auf einer Menge M ist eine Abbildung, die jedem Paar von Elementen aus M ein Element aus M zuweist. Die Addition, Subtraktion und Multiplikation sind uns wohlbekannte Verknüpfungen auf der Menge der reellen Zahlen. Wir haben aber bereits in Definition 2.17 gesehen, dass es oft sinnvoll ist, auch auf anderen Mengen Verknüpfungen zu definieren. Dort haben wir die Verknüpfungen $\oplus_n$ und $\odot_n$ auf der Menge $\mathbb{Z}/n\mathbb{Z}$ betrachtet. In diesem Abschnitt wollen wir uns allgemein mit den Eigenschaften von Verknüpfungen beschäftigen. Dabei abstrahieren wir von der konkreten Grundmenge und der konkreten Verknüpfung und nutzen nur die wesentlichen Eigenschaften aus.

Definition 4.15. *Für eine Menge M nennen wir eine Abbildung $\circ\colon M \times M \to M$ eine* Verknüpfung *auf M. Für $a, b \in M$ schreiben wir statt $\circ(a, b)$ auch $a \circ b$.*

a) $\circ$ ist assoziativ $\iff \forall a, b, c \in M : (a \circ b) \circ c = a \circ (b \circ c)$.

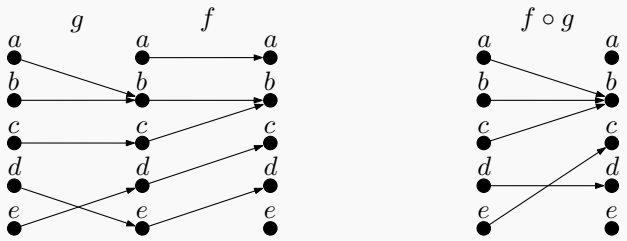
b) $\circ$ ist kommutativ $\iff \forall a, b \in M : a \circ b = b \circ a$.

Beispiele

- Auf $\mathbb{R}$ sind die Addition und Multiplikation sowohl assoziativ als auch kommutativ. Die Subtraktion ist weder assoziativ noch kommutativ, was der Leser sich anhand einfacher Gegenbeispiele klar machen sollte. Die Division ist keine Verknüpfung auf $\mathbb{R}$, da $\frac{x}{0}$ für $x \in \mathbb{R}$ nicht definiert ist.

Auch auf $\mathbb{N}$ sind die Addition und Multiplikation assoziative und kommutative Verknüpfungen. Allerdings handelt es sich weder bei der Subtraktion noch bei der Division um eine Verknüpfung auf $\mathbb{N}$, da nicht jedem Paar von natürlichen Zahlen wieder eine solche zugewiesen wird. So ist zum Beispiel $1 - 2 \notin \mathbb{N}$ und $\frac{1}{2} \notin \mathbb{N}$.

- Auf $\mathbb{R}$ ist die Verknüpfung $\star$ mit $x \star y = x$ für alle $x, y \in \mathbb{R}$ assoziativ aber nicht kommutativ. Die Verknüpfung $\perp$ mit $x \perp y = -x - y$ für alle $x, y \in \mathbb{R}$ ist hingegen kommutativ aber nicht assoziativ. Dies sollte der Leser als Übung begründen.
- Für eine Menge X bezeichnen wir mit $\mathrm{Abb}(X)$ die Menge aller Abbildungen $f\colon X \to X$. Auf dieser Menge definieren wir eine Verknüpfung $\circ$, die zwei Abbildungen $f \in \mathrm{Abb}(X)$ und $g \in \mathrm{Abb}(X)$ die Abbildung zuweist, die entsteht, wenn man g und f nacheinander ausführt. Formal ist die Abbildung $f \circ g\colon X \to X$ für alle $x \in X$ durch $(f \circ g)(x) = f(g(x))$ definiert. Wir nennen $f \circ g$ auch die *Komposition* von f und g. Die folgende Abbildung zeigt ein Beispiel für eine solche Komposition.



Die Verknüpfung $\circ$ ist assoziativ, denn für $f, g, h \in \mathrm{Abb}(X)$ gilt für alle $x \in X$

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

Der Leser überlege sich, dass die Komposition $\circ$ für Mengen X mit mindestens zwei Elementen nicht kommutativ ist.

Aus Definition 4.15 folgt, dass bei einer assoziativen Verknüpfung $\circ$ die Klammern bei der Verknüpfung von drei Elementen weggelassen werden können. Schreiben wir $a \circ b \circ c$, so ist nämlich egal, ob dies als $(a \circ b) \circ c$ oder als $a \circ (b \circ c)$ interpretiert wird. Mithilfe von vollständiger Induktion kann man diese Aussage auch auf Verknüpfungen von endlich vielen Elementen erweitern.

Definition 4.16. *Es sei M eine Menge mit einer Verknüpfung $\circ$. Ein Element $e \in M$ heißt* neutrales Element*, wenn für alle $x \in M$ gilt*

$$e \circ x = x \circ e = x.$$

Zunächst halten wir fest, dass das neutrale Element einer Verknüpfung, sofern es existiert, eindeutig ist. Denn seien $e \in M$ und $e' \in M$ beides neutrale Elemente, so gilt gemäß der obigen Definition $e = e \circ e' = e'$.

Beispiele

- Das neutrale Element der Addition auf $\mathbb{R}$ ist die Null, denn es gilt $x + 0 = 0 + x = x$ für alle $x \in \mathbb{R}$. Das neutrale Element der Multiplikation auf $\mathbb{R}$ ist die Eins, denn es gilt $x \cdot 1 = 1 \cdot x = x$ für alle $x \in \mathbb{R}$. Die Subtraktion auf $\mathbb{R}$ besitzt kein neutrales Element, denn für $x \neq 0$ gibt es kein $e \in \mathbb{R}$ mit $e - x = x - e = x$.
- Für eine Menge X betrachten wir wieder die Menge $\mathrm{Abb}(X)$ mit der Komposition $\circ$. Die Identität id_X mit $\mathrm{id}_X(x) = x$ für alle $x \in X$ ist das neutrale Element der Komposition.

Definition 4.17. *Es sei M eine Menge mit einer Verknüpfung $\circ$ und einem neutralen Element $e \in M$. Für $x \in M$ nennen wir ein Element $y \in M$* inverses Element zu x *oder* Inverses von x*, wenn $x \circ y = y \circ x = e$ gilt. Ein Element $x \in M$ heißt* invertierbar*, wenn es ein inverses Element besitzt.*

Auch hier halten wir zunächst fest, dass inverse Elemente, sofern sie existieren, bei einer assoziativen Verknüpfung eindeutig sind. Sei M eine Menge mit einer assoziativen Verknüpfung $\circ$ und einem neutralen Element $e \in M$ und sei a ein invertierbares Element. Sind b und b' inverse Elemente zu a, so gilt

$$b = e \circ b = (b' \circ a) \circ b = b' \circ (a \circ b) = b' \circ e = b'.$$

Ist eine Menge M mit einer assoziativen Verknüpfung $\circ$ mit einem neutralen Element $e \in M$ gegeben, so bezeichnen wir im Folgenden das Inverse zu einem Element $x \in M$, sofern es existiert, mit x^{-1}. Diese Notation ist zulässig, da das inverse Element im Falle seiner Existenz eindeutig bestimmt ist.

Beispiele

- Für die Addition auf $\mathbb{R}$ ist die Null das neutrale Element. Jedes Element $x \in \mathbb{R}$ besitzt ein inverses Element $y = -x$, denn es gilt $x+(-x) = (-x)+x = 0$. Für die Multiplikation auf $\mathbb{R}$ ist die Eins das neutrale Element. Jedes Element $x \in \mathbb{R}\setminus\{0\}$ besitzt ein inverses Element $y = \frac{1}{x}$, denn es gilt $x \cdot \frac{1}{x} = \frac{1}{x} \cdot x = 1$. Die Null besitzt bezüglich der Multiplikation kein Inverses, denn es gibt kein $x \in \mathbb{R}$ mit $0 \cdot x = x \cdot 0 = 1$.
- Auch für die Addition auf $\mathbb{Z}$ ist die Null das neutrale Element und alle Elemente sind invertierbar. Für die Multiplikation auf $\mathbb{Z}$ ist weiterhin Eins das neutrale Element, allerdings ist kein Element $x \in \mathbb{Z} \setminus \{1, -1\}$ invertierbar, da für solche Elemente $\frac{1}{x} \notin \mathbb{Z}$ gilt.

Für eine Menge X bezeichnen wir mit $\mathrm{Per}(X) \subseteq \mathrm{Abb}(X)$ die Menge aller bijektiven Abbildungen $f\colon X \to X$. Diese Schreibweise ist dadurch motiviert, dass eine bijektive Abbildung einer Menge in sich selbst als Permutation der Menge angesehen werden kann.

Theorem 4.18. *Es sei X eine beliebige Menge. Eine Abbildung $f \in \mathrm{Abb}(X)$ besitzt genau dann ein Inverses bezüglich der Komposition $\circ$, wenn $f \in \mathrm{Per}(X)$ gilt.*

Beweis. Ist $f \in \mathrm{Per}(X)$, so können wir die Umkehrabbildung f^{-1} definieren. Für $y \in X$ gilt dabei $f^{-1}(y) = x$, wobei $x \in X$ das eindeutige Element mit $f(x) = y$ ist. Dass ein solches Element existiert, folgt aus der Surjektivität von f. Dass es eindeutig ist, folgt aus der Injektivität von f. Sei $x \in X$ beliebig und sei $f(x) = y$. Dann gilt per Definition $f^{-1}(y) = x$, woraus $f^{-1}(f(x)) = f^{-1}(y) = x$ folgt. Damit ist $f^{-1} \circ f = \mathrm{id}_X$. Sei nun $y \in X$ beliebig und sei $f^{-1}(y) = x$. Dann gilt per Definition $f(x) = y$, woraus folgt $f(f^{-1}(y)) = f(x) = y$. Damit ist auch $f \circ f^{-1} = \mathrm{id}_X$. Insgesamt ist damit gezeigt, dass jede bijektive Abbildung ein Inverses bezüglich der Komposition besitzt.

Es sei $f \in \mathrm{Abb}(x)$ nicht bijektiv. Ist f nicht injektiv, so gibt es $x, x' \in X$ mit $x \neq x'$ und $f(x) = f(x')$. Angenommen, es gibt ein Inverses $g \in \mathrm{Abb}(X)$ zu f. Dann gilt insbesondere $g(f(x)) = x = \mathrm{id}_X(x)$ und $g(f(x')) = x' = \mathrm{id}_X(x')$. Daraus folgt wegen $f(x) = f(x')$ aber direkt $x = x'$ im Widerspruch zu der Wahl von x und x'. Abbildungen, die nicht injektiv sind, besitzen also keine Inversen.

Sei nun f nicht surjektiv. Dann gibt es ein $y \in X$, für das es kein $x \in X$ mit $f(x) = y$ gibt. Angenommen, es gibt ein Inverses $g \in \mathrm{Abb}(X)$ zu f. Dann gilt insbesondere $f(g(y)) = y$. Wegen $g(y) \in X$ ist dies aber ein Widerspruch dazu, dass es kein $x \in X$ mit $f(x) = y$ gibt. Abbildungen, die nicht surjektiv sind, besitzen also keine Inversen. Somit ist insgesamt gezeigt, dass Abbildungen, die nicht bijektiv sind, keine Inversen besitzen. □

4.3.1 Halbgruppen, Monoide und Gruppen

Bei einer *algebraischen Struktur* handelt es sich um eine Menge, die mit einer oder mehreren Verknüpfungen versehen ist, die bestimmte Eigenschaften erfüllen. In diesem Abschnitt beschäftigen wir uns zunächst mit Mengen, die mit nur einer Verknüpfung versehen sind.

Definition 4.19. *Ist G eine Menge und $\circ$ eine Verknüpfung auf G, so heißen die folgenden drei Eigenschaften* Gruppenaxiome*.*

a) Die Verknüpfung $\circ$ ist assoziativ.

b) Es existiert ein neutrales Element $e \in G$.

c) Jedes Element in G ist invertierbar.

Erfüllt die Verknüpfung $\circ$ das Gruppenaxiom a), so nennen wir $(G, \circ)$ eine Halbgruppe*. Erfüllt die Verknüpfung $\circ$ die Gruppenaxiome a) und b), so nennen wir $(G, \circ)$ ein* Monoid*. Erfüllt die Verknüpfung $\circ$ alle drei Gruppenaxiome, so nennen wir $(G, \circ)$ eine* Gruppe*.*

Ist $(G, \circ)$ eine Gruppe und ist die Verknüpfung $\circ$ zusätzlich kommutativ, so heißt $(G, \circ)$ eine abelsche Gruppe*.* Abelsche Halbgruppen *und* abelsche Monoide *sind analog definiert. Oft wird statt „abelsch“ auch das Wort „kommutativ“ benutzt.*

Der Leser überlege sich, dass beispielsweise $(\mathbb{R},+)$, $(\mathbb{R}\setminus\{0\},\cdot)$, $(\mathbb{Z},+)$ und $(\mathrm{Per}(X),\circ)$ für eine beliebige Menge X Gruppen sind. Unter diesen Beispielen ist $(\mathrm{Per}(X),\circ)$ die einzige Gruppe, die nicht abelsch ist (für X mit $|X|\geq 3$). Die Paare $(\mathbb{R},\cdot)$, $(\mathbb{Z},\cdot)$, $(\mathbb{N},+)$ und $(\mathrm{Abb}(X),\circ)$ für eine beliebige Menge X mit mindestens zwei Elementen sind hingegen keine Gruppen. Bis auf $(\mathbb{N},+)$, was nur eine Halbgruppe ist, sind alle diese Paare Monoide. Auch dies sollte der Leser als Übung begründen.

Gruppen sind ein erster Schritt hin zur Abstraktion von konkreten Grundmengen und Verknüpfungen. Wir können uns nun mit der Frage beschäftigen, welche Rechenregeln allgemein in Gruppen gelten, ohne dabei Bezug auf die konkrete Grundmenge oder Verknüpfung zu nehmen.

Theorem 4.20. *Sei $(G,\circ)$ eine Gruppe. Dann gelten die folgenden Rechenregeln.*

a) Kürzungsregeln: Für alle Elemente $a,x,y\in G$ gilt

$$a\circ x = a\circ y \Rightarrow x=y \quad \textit{und} \quad x\circ a = y\circ a \Rightarrow x=y.$$

b) Eindeutige Lösbarkeit von Gleichungen: Für alle $a,b\in G$ existiert genau ein $x\in G$ mit $a\circ x=b$ und es existiert genau ein $y\in G$ mit $y\circ a=b$.

Beweis. a) Aus $a\circ x = a\circ y$ folgt mithilfe der Gruppenaxiome

$$x = e\circ x = (a^{-1}\circ a)\circ x = a^{-1}\circ(a\circ x) = a^{-1}\circ(a\circ y) = (a^{-1}\circ a)\circ y = e\circ y = y.$$

Die zweite Kürzungsregel folgt analog.

b) Die Elemente $x=a^{-1}\circ b$ und $y=b\circ a^{-1}$ leisten das Gewünschte. Ihre Eindeutigkeit folgt aus den Kürzungsregeln. □

Beispiele: $(\mathbb{Z}/n\mathbb{Z},\oplus_n)$ und $(\mathbb{Z}/n\mathbb{Z},\odot_n)$

Der Leser erinnere sich daran, dass $\mathbb{Z}/n\mathbb{Z}$ die Menge der Äquivalenzklassen der Äquivalenzrelation $\equiv_n$ auf $\mathbb{Z}$ bezeichnet. Es gilt also $\mathbb{Z}/n\mathbb{Z}=\{[\![0]\!],[\![1]\!],[\![2]\!],\ldots,[\![n-1]\!]\}$, wobei $[\![i]\!]$ die Menge aller Zahlen aus $\mathbb{Z}$ beschreibt, die bei Division durch n den Rest i lassen. Aus Theorem 2.18 folgt, dass $[\![i]\!]\oplus_n[\![j]\!]=[\![i+j]\!]$ und $[\![i]\!]\odot_n[\![j]\!]=[\![i\cdot j]\!]$ für alle $i,j\in\mathbb{Z}$ gilt.

Die Assoziativität der Verknüpfung $\oplus_n$ folgt direkt aus der Assoziativität der normalen Addition, denn für $i,j,k\in\mathbb{Z}$ gilt

$$([\![i]\!]\oplus_n[\![j]\!])\oplus_n[\![k]\!] = [\![i+j]\!]\oplus_n[\![k]\!] = [\![i+j+k]\!] = [\![i]\!]\oplus_n[\![j+k]\!] = [\![i]\!]\oplus_n([\![j]\!]\oplus_n[\![k]\!]).$$

Man rechnet leicht nach, dass das neutrale Element die Äquivalenzklasse $[\![0]\!]$ ist und dass das inverse Element zu einer Äquivalenzklasse $[\![i]\!]$ die Klasse $[\![-i]\!]$ ist. Demnach ist $(\mathbb{Z}/n\mathbb{Z},\oplus_n)$ eine Gruppe.

Die Assoziativität der Verknüpfung $\odot_n$ folgt analog aus der Assoziativität der normalen Multiplikation. Ebenso rechnet man leicht nach, dass die Äquivalenzklasse $[\![1]\!]$ das neutrale Element ist. Damit ist $(\mathbb{Z}/n\mathbb{Z},\odot_n)$ ein Monoid. Es ist keine Gruppe, denn die Äquivalenzklasse $[\![0]\!]$ besitzt kein Inverses. Handelt es sich aber bei $(\mathbb{Z}/n\mathbb{Z}\setminus\{[\![0]\!]\},\odot_n)$ um eine Gruppe? Wir werden später sehen, dass dies genau dann der Fall ist, wenn n eine Primzahl ist.

Den Beweis der folgenden wichtigen Aussage überlassen wir dem Leser als Übung.

Theorem 4.21. *Es sei $(M, \circ)$ ein Monoid und es sei $G \subseteq M$ die Menge der invertierbaren Elemente. Dann gilt für alle $x, y \in G$ auch $x \circ y \in G$. Außerdem ist $(G, \star)$ eine Gruppe, wobei $\star\colon G \times G \to G$ die auf G eingeschränkte Verknüpfung $\circ$ bezeichnet, d. h. es gilt $x \star y = x \circ y$ für alle $x, y \in G$.*

4.3.2 Ringe und Körper

Wir beschäftigen uns nun mit Strukturen, die mit zwei Verknüpfungen versehen sind. Diese Verknüpfungen nennen wir im Folgenden $+$ und $\cdot$, sie haben aber im Allgemeinen nichts mit der gewöhnlichen Addition und Multiplikation zu tun.

Definition 4.22. *Es sei R eine Menge mit zwei Verknüpfungen $+$ und $\cdot$. Wir nennen $(R, +, \cdot)$ einen* Ring*, wenn die folgenden Eigenschaften erfüllt sind.*

a) $(R, +)$ ist eine abelsche Gruppe.

b) Die Verknüpfung $\cdot$ ist assoziativ.

c) Es gelten die Distributivgesetze*, das heißt für alle $a, b, c \in R$ gilt*

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad \textit{und} \quad (a + b) \cdot c = (a \cdot c) + (b \cdot c).$$

Ist die Verknüpfung $\cdot$ zusätzlich kommutativ, so heißt $(R, +, \cdot)$ kommutativer Ring*. Besitzt die Halbgruppe $(R, \cdot)$ ein neutrales Element, so heißt $(R, +, \cdot)$* Ring mit Eins *und wir nennen $x \in R$* invertierbar *oder* Einheit*, wenn x bezüglich der Verknüpfung $\cdot$ invertierbar ist. Wir bezeichnen in diesem Fall mit $R^* \subseteq R$ die Menge der Einheiten. Gemäß Theorem 4.21 handelt es sich bei $(R^*, \cdot)$ um eine Gruppe. Diese nennen wir auch die* Einheitengruppe *von R.*

Um Klammern zu sparen, vereinbaren wir, dass wie bei der normalen Addition und Multiplikation Punkt- vor Strichrechnung gilt. Außerdem bezeichnen wir in einem Ring $(R, +, \cdot)$ mit Eins das neutrale Element bezüglich der Verknüpfung $+$ mit 0 und das neutrale Element bezüglich der Verknüpfung $\cdot$ mit 1. Wir weisen aber noch einmal ausdrücklich darauf hin, dass dies genauso wie die Bezeichnungen $+$ und $\cdot$ nur als Schreibweise zu verstehen ist. Im Allgemeinen haben 0 und 1 nichts mit den entsprechenden Zahlen aus $\mathbb{R}$ zu tun. Ebenso bezeichnen wir für $x \in R$ mit $-x$ das eindeutige inverse Element zu x bezüglich der Verknüpfung $+$ und mit x^{-1} das eindeutige inverse Element zu x bezüglich der Verknüpfung $\cdot$, sofern es existiert.

Beispiele

- Bei $(\mathbb{Z}, +, \cdot)$ handelt es sich um einen kommutativen Ring, wobei $+$ und $\cdot$ hier die herkömmliche Addition und Multiplikation bezeichnen. Für diesen Ring gilt $\mathbb{Z}^* = \{1, -1\}$. Ebenso handelt es sich bei $(\mathbb{R}, +, \cdot)$ um einen kommutativen Ring mit $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$.

- Der Leser überlege sich, dass es sich auch bei $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ um einen kommutativen Ring handelt. Das neutrale Element bezüglich $\oplus_n$ ist $[\![0]\!]$ und das neutrale Element bezüglich $\odot_n$ ist $[\![1]\!]$. Wir betrachten als Beispiel die *Verknüpfungstafeln* für $n = 3$. Eine Verknüpfungstafel ist eine Tabelle, die für jedes Paar von Elementen das Ergebnis der Verknüpfung angibt.

$\oplus_3$	$[\![0]\!]$	$[\![1]\!]$	$[\![2]\!]$
$[\![0]\!]$	$[\![0]\!]$	$[\![1]\!]$	$[\![2]\!]$
$[\![1]\!]$	$[\![1]\!]$	$[\![2]\!]$	$[\![0]\!]$
$[\![2]\!]$	$[\![2]\!]$	$[\![0]\!]$	$[\![1]\!]$

$\odot_3$	$[\![0]\!]$	$[\![1]\!]$	$[\![2]\!]$
$[\![0]\!]$	$[\![0]\!]$	$[\![0]\!]$	$[\![0]\!]$
$[\![1]\!]$	$[\![0]\!]$	$[\![1]\!]$	$[\![2]\!]$
$[\![2]\!]$	$[\![0]\!]$	$[\![2]\!]$	$[\![1]\!]$

  In diesem Beispiel gilt $(\mathbb{Z}/3\mathbb{Z})^* = \{[\![1]\!], [\![2]\!]\}$, d. h. alle Elemente außer dem neutralen Element der Addition besitzen ein inverses Element bezüglich $\odot_3$. Dies ist aber nicht für jedes $n \in \mathbb{N}$ der Fall, wie wir später noch sehen werden.

Kommutative Ringe mit Eins, in denen jedes Element außer der 0 invertierbar ist, haben eine große Bedeutung in der Mathematik, deshalb gibt es auch eine eigene Bezeichnung für Ringe mit dieser Eigenschaft.

Definition 4.23. *Ist $(R, +, \cdot)$ ein Ring und ist $(R \setminus \{0\}, \cdot)$ eine abelsche Gruppe, so nennen wir $(R, +, \cdot)$ einen* Körper.

Man beachte, dass aus dieser Definition direkt folgt, dass ein Körper mindestens zwei Elemente enthält, denn zusätzlich zum neutralen Element der Addition 0 muss es ein neutrales Element der Multiplikation 1 geben, da sonst $(R \setminus \{0\}, \cdot)$ keine Gruppe wäre. Insbesondere gilt also in jedem Körper $0 \neq 1$.

Beispiele

- Bei $(\mathbb{Q}, +, \cdot)$ und $(\mathbb{R}, +, \cdot)$ handelt es sich um Körper, nicht aber bei $(\mathbb{Z}, +, \cdot)$.
- Bei $(\mathbb{Z}/3\mathbb{Z}, \oplus_3, \odot_3)$ handelt es sich um einen Körper wie die obigen Verknüpfungstafeln zeigen. Der Leser sollte als Übung die Verknüpfungstafeln für $(\mathbb{Z}/4\mathbb{Z}, \oplus_4, \odot_4)$ aufstellen und argumentieren, dass es sich dabei nicht um einen Körper handelt.

Wir zeigen nun die oben bereits angesprochene Aussage, dass $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ genau dann ein Körper ist, wenn n eine Primzahl ist. Dazu benötigen wir das folgende Lemma, das wir hier nicht beweisen, da es aus einer allgemeineren Aussage folgt, die wir später beweisen werden.

Lemma 4.24. *Sind $a, b \in \mathbb{Z}$ teilerfremd, dann gibt es $x, y \in \mathbb{Z}$ mit $ax + by = 1$.*

Außerdem benötigen wir noch das folgende Lemma.

Lemma 4.25. *Es sei $(R, +, \cdot)$ ein Körper. Dann besitzt das neutrale Element der Addition 0 kein multiplikatives Inverses.*

Beweis. Es gibt kein multiplikatives Inverses $a \in R$ mit $a \cdot 0 = 1$, da $a \cdot 0 = 0 \neq 1$ für jedes $a \in R$ gilt. Die Gültigkeit der Aussage $a \cdot 0 = 0$ für jedes $a \in R$ mag dem Leser vielleicht offensichtlich erscheinen, man bedenke aber, dass 0 und $\cdot$ nur Platzhalter sind, die im Allgemeinen nichts mit der Zahl Null und der normalen Multiplikation zu tun haben. Wir können lediglich auf die bekannten Rechenregeln in Körpern zurückgreifen, um die Aussage zu begründen. Für $a \in R$ gilt

$$\begin{aligned}
a \cdot 0 &= a \cdot 0 + 0 && \text{(neutrales Element der Addition)} \\
&= a \cdot 0 + (a \cdot 0 + (-(a \cdot 0))) && \text{(inverses Element der Addition)} \\
&= (a \cdot 0 + a \cdot 0) + (-(a \cdot 0)) && \text{(Assoziativität der Addition)} \\
&= a \cdot (0 + 0) + (-(a \cdot 0)) && \text{(Distributivgesetz)} \\
&= a \cdot 0 + (-(a \cdot 0)) && \text{(neutrales Element der Addition)} \\
&= 0. && \text{(inverses Element der Addition)}
\end{aligned}$$

Damit ist der Beweis abgeschlossen. □

Theorem 4.26. *Der Ring $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ ist genau dann ein Körper, wenn n eine Primzahl ist.*

Beweis. Wir wissen bereits, dass es sich bei $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ für jedes $n \in \mathbb{N}$ um einen kommutativen Ring mit Eins handelt. Somit müssen wir nur noch die Frage betrachten, welche Elemente ein Inverses bezüglich $\odot_n$ besitzen.

Wir betrachten nun zunächst den Fall, dass es sich bei $n = p$ um eine Primzahl handelt. In diesem Fall zeigen wir, dass jedes Element in $\mathbb{Z}/p\mathbb{Z} \setminus \{[\![0]\!]\} = \{[\![1]\!], [\![2]\!], \ldots, [\![p-1]\!]\}$ invertierbar bezüglich $\odot_p$ ist. Sei $a \in \{1, \ldots, p-1\}$ beliebig. Da p eine Primzahl ist, haben a und p keinen gemeinsamen Teiler außer 1. Gemäß Lemma 4.24, gibt es $x, y \in \mathbb{Z}$ mit $ax + py = 1$. Es gilt dann

$$\begin{aligned}
[\![a]\!] \odot_p [\![x]\!] &= ([\![a]\!] \odot_p [\![x]\!]) \oplus_p [\![0]\!] \\
&= ([\![a]\!] \odot_p [\![x]\!]) \oplus_p [\![py]\!] \\
&= [\![ax]\!] \oplus_p [\![py]\!] \\
&= [\![ax + py]\!] = [\![1]\!],
\end{aligned}$$

wobei wir bei der zweiten Gleichung ausgenutzt haben, dass py den Rest 0 bei Division durch p lässt, woraus $[\![py]\!] = [\![0]\!]$ folgt. Die obige Rechnung besagt, dass die Äquivalenzklasse $[\![x]\!]$ invers zu der Äquivalenzklasse $[\![a]\!]$ ist. Damit ist gezeigt, dass jedes Element aus $\mathbb{Z}/p\mathbb{Z} \setminus \{[\![0]\!]\}$ ein multiplikatives Inverses besitzt. Somit ist $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ ein Körper.

Es bleibt, den Fall zu betrachten, dass n keine Primzahl ist. Für $n = 1$ gilt $|\mathbb{Z}/n\mathbb{Z}| = 1$. Da jeder Körper mindestens zwei Elemente enthält, handelt es sich für $n = 1$ bei $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ nicht um einen Körper. Sei nun $n > 1$ keine Primzahl. Dann gibt es zwei Zahlen $a, b \in \{2, 3, 4, \ldots, n-1\}$ mit $n = a \cdot b$. Für diese Zahlen gilt $[\![a]\!] \neq [\![0]\!]$ und $[\![b]\!] \neq [\![0]\!]$, aber $[\![a]\!] \odot_n [\![b]\!] = [\![a \cdot b]\!] = [\![0]\!]$. Gemäß Lemma 4.25 ist $[\![0]\!]$ nicht invertierbar. Wären jedoch sowohl $[\![a]\!]$ als auch $[\![b]\!]$ invertierbar mit den entsprechenden

Inversen $[\![a]\!]^{-1}$ und $[\![b]\!]^{-1}$, so wäre auch $[\![a]\!] \odot_n [\![b]\!] = [\![0]\!]$ invertierbar mit dem Inversen $[\![b]\!]^{-1} \odot_n [\![a]\!]^{-1}$. Daraus folgt, dass $[\![a]\!]$ und $[\![b]\!]$ nicht beide invertierbar sein können. Es gibt somit ein Element aus $\mathbb{Z}/n\mathbb{Z} \setminus \{[\![0]\!]\}$, das nicht invertierbar ist. Demzufolge ist $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ kein Körper. □

4.3.3 Euklidischer Algorithmus

In einigen Anwendungen wie zum Beispiel der Kryptographie benötigt man effiziente Algorithmen, um mit sehr großen Zahlen zu rechnen. Wir lernen in diesem Abschnitt einige Grundlagen für solche Algorithmen kennen und werden nebenbei auch Lemma 4.24 aus dem vorangegangenen Abschnitt beweisen.

Zunächst beschäftigen wir uns damit, den größten gemeinsamen Teiler zweier Zahlen zu berechnen. Diesen Begriff haben wir in Kapitel 2 schon benutzt, ohne ihn formal zu definieren. Das holen wir jetzt nach.

Definition 4.27. *Es seien $d \in \mathbb{Z}$ und $x \in \mathbb{Z}$. Wir nennen d einen* Teiler *von x, wenn es ein $k \in \mathbb{Z}$ mit $dk = x$ gibt. Wir schreiben dann $d \mid x$ und sagen auch, dass x durch d* teilbar *ist.*

Für zwei ganze Zahlen $x, y \in \mathbb{Z}$ heißt $d \in \mathbb{N}$ größter gemeinsamer Teiler $\mathrm{ggT}(x, y)$ *von x und y, wenn die folgenden beiden Bedingungen erfüllt sind.*

a) Die Zahl d ist ein gemeinsamer Teiler von x und y, d. h. $d \mid x$ und $d \mid y$.

b) Jeder gemeinsame Teiler d' von x und y ist auch ein Teiler von d, also

$$\forall d' \in \mathbb{Z} : ((d' \mid x) \wedge (d' \mid y)) \Rightarrow d' \mid d.$$

Wir werden gleich argumentieren, dass es stets einen größten gemeinsamer Teiler gemäß der obigen Definition gibt. Zusätzlich sollte der Leser sich überlegen, dass aus der Definition direkt hervorgeht, dass der größte gemeinsame Teiler zweier Zahlen $x, y \in \mathbb{Z}$ wegen der Bedingung $d \in \mathbb{N}$ eindeutig bestimmt ist, wenn er existiert. Wir bezeichnen ihn im Folgenden mit $\mathrm{ggT}(x, y)$.

Definition 4.28. *Zwei Zahlen $x, y \in \mathbb{Z}$ heißen* teilerfremd*, wenn $\mathrm{ggT}(x, y) = 1$ gilt.*

Die Existenz eines größten gemeinsamen Teilers weisen wir konstruktiv nach. Das bedeutet, wir geben einen Algorithmus an, der einen solchen Teiler berechnet. Dieser sogenannte *euklidische Algorithmus* kann leicht in jeder gängigen Programmiersprache implementiert werden und berechnet auch für sehr große Zahlen schnell den größten gemeinsamen Teiler. Wir beschreiben den Algorithmus in *Pseudocode*. Das bedeutet, wir geben den Algorithmus in keiner konkreten Programmiersprache an, beschreiben ihn aber so detailliert, dass er von einem Programmierer direkt in richtigen Quelltext übertragen werden kann.

Euklid($x_0 \in \mathbb{Z}, x_1 \in \mathbb{Z}$)

1. **if** ($x_0 < x_1$) { vertausche x_0 und x_1; }
2. $i := 1$;
3. **while** ($x_i \neq 0$) {
4. $\quad x_{i+1} := x_{i-1} \bmod x_i$;
5. $\quad i := i + 1$;
6. }
7. **return** x_{i-1};

In dem Pseudocode wird eine Folge $x_0, x_1, x_2, \ldots$ von Variablen generiert. Man kann den Algorithmus zwar auch so abändern, dass er mit drei Variablen auskommt (was man bei einer tatsächlichen Implementierung auch tun würde), die Folge der x_i erleichtert es uns aber, den Algorithmus zu analysieren. Die Anweisung in der vierten Zeile steht für eine Division mit Rest. Dabei wird x_{i-1} durch x_i geteilt und der Rest wird in x_{i+1} geschrieben. Dieser Rest stammt stets aus der Menge $\{0, 1, 2, \ldots, |x_i| - 1\}$.

Wir betrachten den euklidischen Algorithmus zunächst für das Beispiel $x_0 = 1365$ und $x_1 = 510$. In diesem Beispiel wird die Sequenz

$$(x_0, x_1, \ldots, x_5) = (1365, 510, 345, 165, 15, 0)$$

generiert und $x_4 = 15$ wird ausgegeben.

Theorem 4.29. *Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler von x_0 und x_1.*

Beweis. Wir können ohne Beschränkung der Allgemeinheit davon ausgehen, dass $x_0 \geq x_1$ gilt, da ansonsten im ersten Schritt x_0 und x_1 vertauscht werden. Gilt $x_i \neq 0$, so berechnet der Algorithmus ein $x_{i+1} \in \{0, 1, 2, \ldots, |x_i| - 1\}$ mit

$$x_{i-1} = q_i \cdot x_i + x_{i+1}.$$

für eine geeignete Zahl $q_i \in \mathbb{Z}$. Daraus können wir zunächst ableiten, dass der Algorithmus immer terminiert, denn es gilt $x_i \geq 0$ für alle $i \geq 2$ und $x_{i+1} < |x_i|$ für $i \geq 0$ Dies bedeutet insgesamt, dass $|x_1| > x_2 > x_3 > x_4 > \ldots \geq 0$ gilt. Somit erreicht der Algorithmus nach höchstens $|x_1|$ vielen Iterationen einen Index $n + 1$ mit $x_{n+1} = 0$. Er terminiert dann mit der Ausgabe x_n.

Wir weisen nach, dass die Zahl x_n die beiden Eigenschaften aus Definition 4.27 erfüllt. Zunächst weisen wir nach, dass x_n ein gemeinsamer Teiler von x_0 und x_1 ist. Dies folgt mit einem induktiven Argument. Zunächst ist klar, dass x_n ein Teiler von x_{n-1} ist, denn es gilt

$$x_{n-1} = q_n \cdot x_n + x_{n+1} = q_n \cdot x_n + 0 = q_n \cdot x_n.$$

Daraus folgt, dass x_n auch ein Teiler von x_{n-2} ist, denn es gilt

$$x_{n-2} = q_{n-1} \cdot x_{n-1} + x_n$$

und die Summe zweier durch x_n teilbarer Zahlen ist ebenfalls durch x_n teilbar. Daraus folgt wiederum, dass x_n ein Teiler von x_{n-3} ist, denn es gilt

$$x_{n-3} = q_{n-2} \cdot x_{n-2} + x_{n-1}$$

und sowohl x_{n-2} als auch x_{n-1} sind durch x_n teilbar. Dieses Argument können wir solange fortsetzen, bis wir zu der Schlussfolgerung kommen, dass auch x_0 und x_1 durch x_n teilbar sind.

Nun weisen wir nach, dass jeder gemeinsame Teiler d' von x_0 und x_1 auch x_n teilt. Auch dazu nutzen wir ein induktives Argument. Da d' ein Teiler von x_0 und x_1 ist, ist es auch ein Teiler von

$$x_2 = x_0 - q_1 \cdot x_1.$$

Demzufolge ist es auch ein Teiler von

$$x_3 = x_1 - q_2 \cdot x_2.$$

und so weiter. Setzen wir dieses Argument fort, so erhalten wir, dass d' ein Teiler von x_n ist. □

Wir beweisen nun das folgende Lemma, aus dem Lemma 4.24 direkt als Spezialfall folgt.

Lemma 4.30. *Sind $a, b \in \mathbb{Z}$ und $d = \mathrm{ggT}(a, b)$, dann gibt es $x, y \in \mathbb{Z}$ mit $ax + by = d$.*

Beweis. Dieses Lemma ergibt sich aus dem euklidischen Algorithmus, mit dem der größte gemeinsame Teiler d von $x_0 = a$ und $x_1 = b$ berechnet werden kann. Im Beweis von Theorem 4.29 haben wir argumentiert, dass es ein n gibt, für das $x_n = d$ gilt. Außerdem gibt es $q_1, q_2, \ldots, q_n \in \mathbb{Z}$, für die gilt

$$\begin{aligned} d = x_n &= x_{n-2} - q_{n-1} \cdot x_{n-1} \\ &= x_{n-2} - q_{n-1} \cdot (x_{n-3} - q_{n-2} \cdot x_{n-2}) \\ &= -q_{n-1} \cdot x_{n-3} + (1 + q_{n-1}q_{n-2}) \cdot x_{n-2}. \end{aligned}$$

Setzen wir diese Rechnung fort und ersetzen als nächstes x_{n-2} durch $x_{n-4} - q_{n-3} \cdot x_{n-3}$, danach x_{n-3} durch $x_{n-5} - q_{n-4} \cdot x_{n-4}$ und so weiter, so erhalten wir zwei Zahlen $x, y \in \mathbb{Z}$, für die die Gleichung

$$d = x_n = x \cdot x_0 + y \cdot x_1$$

gilt. Dabei sind $x \in \mathbb{Z}$ und $y \in \mathbb{Z}$ Ausdrücke, die sich aus den q_i zusammensetzen. □

Wir betrachten noch einmal das obige Beispiel mit $x_0 = 1365$ und $x_1 = 510$. Wir erhalten die folgenden Gleichungen:

$$\begin{aligned} 1365 &= 2 \cdot 510 + 345 \\ 510 &= 1 \cdot 345 + 165 \\ 345 &= 2 \cdot 165 + 15 \\ 165 &= 11 \cdot 15 + 0. \end{aligned}$$

Um den größten gemeinsamen Teiler 15 darzustellen, stellen wir die Gleichungen um und setzen sie ineinander ein. Wir erhalten

$$\begin{aligned} 15 &= 345 - 2 \cdot 165 \\ &= 345 - 2 \cdot (510 - 1 \cdot 345) = -2 \cdot 510 + 3 \cdot 345 \\ &= -2 \cdot 510 + 3 \cdot (1365 - 2 \cdot 510) = 3 \cdot 1365 - 8 \cdot 510. \end{aligned}$$

Die gesuchten Koeffizienten sind also 3 und -8.

Bei einem Algorithmus ist es nicht nur wichtig, dass er terminiert und das korrekte Ergebnis liefert, sondern von ganz großer Bedeutung ist auch seine Laufzeit. In kryptographischen Anwendungen kommt es nicht selten vor, dass mit Zahlen gerechnet wird, die über 1000 Dezimalstellen besitzen. Auch für solche Zahlen sollte der euklidische Algorithmus in möglichst wenigen Schritten den größten gemeinsamen Teiler berechnen. Wir konzentrieren uns in dem folgenden Theorem der Einfachheit halber auf den Fall, dass $x_0, x_1 \in \mathbb{N}$ mit $x_0 \geq x_1$ gilt. Negative Zahlen und Eingaben mit $x_0 < x_1$ können aber analog analysiert werden.

Theorem 4.31. *Bei der Eingabe $x_0, x_1 \in \mathbb{N}$ mit $x_0 \geq x_1$, beträgt die Anzahl an Durchläufen der while-Schleife im euklidischen Algorithmus maximal $2 \cdot \log_2(x_0)$.*

Beweis. Wir verwenden wieder die Notationen aus dem Beweis von Theorem 4.29. Wegen $x_0, x_1 \in \mathbb{N}$ sind alle weiteren x_i und auch alle q_i nichtnegativ. Es gilt außerdem $x_0 \geq x_1 > x_2 > x_3 > \ldots > x_n > x_{n+1} = 0$. Für jedes $i \geq 1$ gilt

$$x_{i-1} = q_i \cdot x_i + x_{i+1}$$

und außerdem gilt $q_i \geq 1$ wegen $x_{i-1} \geq x_i > x_{i+1}$. Daraus folgt für jedes $i \geq 1$

$$x_{i-1} \geq x_i + x_{i+1} > 2x_{i+1}.$$

Für jedes $i \geq 1$ ist x_{i-1} demnach mehr als doppelt so groß wie x_{i+1}. Daraus ergibt sich, dass

$$x_{2k} < \frac{x_0}{2^k}$$

für jedes $k \in \mathbb{N}$ gilt. Für $k \geq \log_2(x_0)$ gilt somit $x_{2k} < 1$, also $x_{2k} = 0$. Es gilt somit $n < 2 \cdot \log_2(x_0)$ und damit ist auch die Anzahl der Durchläufe der while-Schleife kleiner als $2 \cdot \log_2(x_0)$. □

Die Anzahl der Schleifendurchläufe wächst beim euklidischen Algorithmus also nur logarithmisch in x_0. Dies ist von großer Bedeutung, denn es ermöglicht uns, den größten gemeinsamen Teiler von sehr großen Zahlen zu berechnen. Ist x_0 beispielsweise höchstens 2^{4096}, so genügen dem euklidischen Algorithmus weniger als 8192 Durchläufe der Schleife. Das kann mit einem modernen PC schnell bewerkstelligt werden.

Nun stellt sich noch die Frage, wie viele Rechenoperationen in jedem Durchlauf der while-Schleife durchgeführt werden. Man könnte auf den ersten Blick behaupten, dass es nur zwei Operationen sind: eine Division mit Rest in Schritt 4 und das Inkrementieren des Zählers in Schritt 5. Da wir aber mit potentiell sehr großen Zahlen hantieren,

die nicht in ein Register passen, ist es zu optimistisch, Schritt 4 mit nur einer Rechenoperation abzuschätzen. Wir möchten hier jedoch nicht weiter auf diese Thematik eingehen, sondern verweisen auf die Vorlesung „Algorithmen und Berechnungskomplexität I“ im dritten Semester, in der wir uns ausführlich mit der Analyse von Laufzeiten von Algorithmen beschäftigen werden.

4.3.4 Chinesischer Restsatz

Bereits in der Schule beschäftigt man sich mit der Lösung von Gleichungssystemen über den reellen Zahlen. In vielen Anwendungen in der Informatik spielen Gleichungssysteme eine Rolle, die nicht über den reellen Zahlen, sondern über anderen Körpern definiert sind. Dabei sind für Primzahlen n insbesondere die Körper $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ von Interesse.

Wir betrachten zunächst ein einfaches Gleichungssystem mit einer Variablen $x \in \mathbb{Z}$ und zwei Gleichungen. Die Variable soll so gewählt werden, dass sie das folgende *Kongruenzsystem* für gegebene $a, b \in \mathbb{Z}$ und $n, m \in \mathbb{N}$ mit $\mathrm{ggT}(n, m) = 1$ löst:

$$\begin{aligned} x &\equiv a \bmod n \\ x &\equiv b \bmod m. \end{aligned} \tag{4.1}$$

Dabei ist $x \equiv a \bmod n$ eine andere Schreibweise für $x \equiv_n a$. Das heißt, die Zahlen x und a lassen denselben Rest bei Division durch n. Da wir in dem Gleichungssystem sowohl Äquivalenzklassen von $\equiv_m$ als auch von $\equiv_n$ betrachten, präzisieren wir unsere Schreibweise ein wenig. Im Folgenden steht $[\![x]\!]_k$ für die Äquivalenzklasse von x bezüglich $\equiv_k$. Die Menge $[\![x]\!]_k$ enthält also alle ganzen Zahlen, die bei Division durch k denselben Rest lassen wie x.

Auf den ersten Blick erscheint das obige Kongruenzsystem sehr speziell zu sein, solche und ähnliche Systeme treten aber tatsächlich oft auf. Es stellen sich direkt einige wichtige Fragen: Gibt es stets eine Lösung? Falls ja, ist sie eindeutig und kann sie effizient gefunden werden? Wir beginnen mit dem folgenden Resultat, das als *chinesischer Restsatz* bekannt ist.

Theorem 4.32. *Für alle $a, b \in \mathbb{Z}$ und $n, m \in \mathbb{N}$ mit $\mathrm{ggT}(n, m) = 1$ gibt es genau eine Lösung $x \in \{0, 1, 2, \ldots, nm - 1\}$ für das Kongruenzsystem* (4.1).

Beweis. Wir definieren eine Funktion

$$\varphi \colon \mathbb{Z}/nm\mathbb{Z} \to (\mathbb{Z}/n\mathbb{Z}) \times (\mathbb{Z}/m\mathbb{Z}) \quad \text{mit} \quad [\![k]\!]_{nm} \mapsto ([\![k]\!]_n, [\![k]\!]_m)$$

für $k \in \{0, 1, \ldots, nm - 1\}$. Die Aussage, dass es eine Lösung $x \in \{0, 1, 2, \ldots, nm - 1\}$ für das Kongruenzsystems gibt, entspricht der Aussage, dass es ein x mit $\varphi([\![x]\!]_{nm}) = ([\![a]\!]_n, [\![b]\!]_m)$ gibt.

Zunächst zeigen wir, dass φ injektiv ist. Seien dazu $k, \ell \in \{0, 1, \ldots, nm - 1\}$ gegeben. Es gilt

$$\varphi([\![k]\!]_{nm}) = \varphi([\![\ell]\!]_{nm}) \iff ([\![k]\!]_n, [\![k]\!]_m) = ([\![\ell]\!]_n, [\![\ell]\!]_m)$$

$$\begin{aligned}
&\iff (\llbracket k \rrbracket_n = \llbracket \ell \rrbracket_n) \wedge (\llbracket k \rrbracket_m = \llbracket \ell \rrbracket_m) \\
&\iff (n \mid (k-\ell)) \wedge (m \mid (k-\ell)) \\
&\implies nm \mid (k-\ell) \\
&\iff \llbracket k \rrbracket_{nm} = \llbracket \ell \rrbracket_{nm}.
\end{aligned}$$

Für die Implikation in der vorletzten Zeile haben wir ausgenutzt, dass die Zahlen n und m teilerfremd sind. Dass diese Implikation korrekt ist, sollte der Leser als Übung beweisen. Recht anschaulich kann man den Beweis führen, indem man sich die Zerlegung von n, m und $k-\ell$ in Primfaktoren anschaut. Insgesamt folgt aus der obigen Rechnung, dass die Funktion φ injektiv ist.

Aus der Injektivität folgt, dass es höchstens eine Lösung $x \in \{0, 1, 2, \ldots, nm-1\}$ des Kongruenzsystems gibt, denn für $x, y \in \{0, 1, 2, \ldots, nm-1\}$ mit $\varphi(\llbracket x \rrbracket_{nm}) = \varphi(\llbracket y \rrbracket_{nm}) = (\llbracket a \rrbracket_n, \llbracket b \rrbracket_m)$ folgt aus der Injektivität $\llbracket x \rrbracket_{nm} = \llbracket y \rrbracket_{nm}$. Da x und y zwischen 0 und $nm-1$ liegen, ist dies gleichbedeutend mit $x = y$.

Um nachzuweisen, dass es stets eine Lösung des Kongruenzsystems gibt, genügt es zu zeigen, dass die Funktion φ surjektiv ist, denn das bedeutet insbesondere, dass es ein $x \in \{0, 1, 2, \ldots, nm-1\}$ mit $\varphi(\llbracket x \rrbracket_{nm}) = (\llbracket a \rrbracket_n, \llbracket b \rrbracket_m)$ gibt. Die Surjektivität folgt bei der Funktion φ interessanterweise direkt aus der Injektivität, denn es gilt

$$|\mathbb{Z}/nm\mathbb{Z}| = |(\mathbb{Z}/n\mathbb{Z}) \times (\mathbb{Z}/m\mathbb{Z})| = nm.$$

Das bedeutet, die Definitions- und die Zielmenge haben dieselbe endliche Kardinalität. Der Leser überlege sich, dass für solche Funktionen die Begriffe injektiv, surjektiv und bijektiv äquivalent sind. Das bedeutet, die Funktion φ ist nicht nur injektiv, sondern auch surjektiv. Damit folgt insgesamt, dass es genau eine Zahl $x \in \{0, 1, 2, \ldots, nm-1\}$ mit $\varphi(\llbracket x \rrbracket_{nm}) = (\llbracket a \rrbracket_n, \llbracket b \rrbracket_m)$ gibt. □

Der obige Beweis ist nicht konstruktiv. Das bedeutet, wir haben nur die Existenz einer Lösung gezeigt, aber keinen Algorithmus angegeben, um eine solche Lösung effizient zu finden. Das holen wir nach und geben im Folgenden noch einen konstruktiven Beweis für die Existenz einer Lösung des Kongruenzsystems an.

Alternativer Beweis für die Existenz einer Lösung des Kongruenzsystems. Wegen der Teilerfremdheit von n und m können wir mithilfe des euklidischen Algorithmus gemäß Lemma 4.30 zwei Zahlen $y, z \in \mathbb{Z}$ mit $ny+mz = 1$ berechnen. Wir setzen $x = bny+amz$ und behaupten, dass dieses x das Kongruenzsystem löst. Zunächst wissen wir, dass

$$1 = ny + mz \equiv mz \bmod n \quad \text{und} \quad 1 = ny + mz \equiv ny \bmod m$$

gilt. In der uns vertrauten Schreibweise können wir diese Eigenschaften als

$$\llbracket 1 \rrbracket_n = \llbracket ny + mz \rrbracket_n = \llbracket mz \rrbracket_n \quad \text{und} \quad \llbracket 1 \rrbracket_m = \llbracket ny + mz \rrbracket_m = \llbracket ny \rrbracket_m$$

ausdrücken. Daraus ergibt sich, wie gewünscht,

$$\llbracket x \rrbracket_n = \llbracket bny + amz \rrbracket_n = \llbracket amz \rrbracket_n = \llbracket a \rrbracket_n \odot_n \llbracket mz \rrbracket_n = \llbracket a \rrbracket_n \odot_n \llbracket 1 \rrbracket_n = \llbracket a \rrbracket_n$$

und

$$[\![x]\!]_m = [\![bny + amz]\!]_m = [\![bny]\!]_m = [\![b]\!]_m \odot_m [\![ny]\!]_m = [\![b]\!]_m \odot_m [\![1]\!]_m = [\![b]\!]_m.$$

Damit haben wir explizit eine Lösung des Kongruenzsystems konstruiert. □

Zur Berechnung einer Lösung des Kongruenzsystems genügt es also, den euklidischen Algorithmus anzuwenden, um die Zahlen y und z mit $ny + mz = 1$ zu bestimmen. Die Laufzeit liegt somit in derselben Größenordnung wie die des euklidischen Algorithmus.

Betrachten wir als Beispiel das Kongruenzsystem

$$\begin{aligned} x &\equiv 3 \bmod 20 \\ x &\equiv 5 \bmod 153. \end{aligned} \tag{4.2}$$

Wir wenden zunächst den euklidischen Algorithmus an und erhalten mit $x_0 = 153$ und $x_1 = 20$ die Sequenz $(x_0, x_1, \ldots, x_6) = (153, 20, 13, 7, 6, 1, 0)$ und die Gleichungen

$$\begin{aligned} 153 &= 7 \cdot 20 + 13 \\ 20 &= 1 \cdot 13 + 7 \\ 13 &= 1 \cdot 7 + 6 \\ 7 &= 1 \cdot 6 + 1 \\ 6 &= 6 \cdot 1 + 0. \end{aligned}$$

Durch Umformen dieser Gleichungen und Einsetzen erhält man

$$\begin{aligned} 1 &= 7 - 1 \cdot 6 \\ &= 7 - 1 \cdot (13 - 1 \cdot 7) = -1 \cdot 13 + 2 \cdot 7 \\ &= -1 \cdot 13 + 2 \cdot (20 - 1 \cdot 13) = 2 \cdot 20 - 3 \cdot 13 \\ &= 2 \cdot 20 - 3 \cdot (153 - 7 \cdot 20) = -3 \cdot 153 + 23 \cdot 20. \end{aligned}$$

Mit $y = 23$ und $z = -3$ gilt also $y \cdot 20 + z \cdot 153 = 1$. Wir setzen demnach

$$x = 5 \cdot y \cdot 20 + 3 \cdot z \cdot 153 = 923.$$

Der Leser sollte überprüfen, dass dieses x wirklich die beiden Kongruenzen (4.2) erfüllt.

Man kann den chinesischen Restsatz auch auf Systeme mit mehr als zwei Kongruenzen erweitern, solange Kongruenzen modulo $n_1, n_2, \ldots, n_r$ mit paarweise teilerfremden n_i betrachtet werden. Analog kann man zeigen, dass für jedes solche Kongruenzsystem

$$\begin{aligned} x &\equiv a_1 \bmod n_1 \\ &\vdots \\ x &\equiv a_r \bmod n_r \end{aligned}$$

eine eindeutige Lösung $x \in \{0, 1, 2, \ldots, N-1\}$ mit $N = n_1 \cdot \ldots \cdot n_r$ existiert.

Anwendungen dieser Erweiterung finden sich zum Beispiel in der Kryptographie. Angenommen, wir möchten ein Geheimnis $S \in \mathbb{N}$ so auf k Personen aufteilen, dass sie

gemeinsam das Geheimnis entschlüsseln können, aber keine Gruppe von $k-1$ Personen alleine das Geheimnis entschlüsseln kann. Dazu wählen wir paarweise teilerfremde Zahlen $n_1, \ldots, n_k$ mit $n_1 \cdot \ldots \cdot n_k > S$ und teilen Person i die Zahl $a_i = S \bmod n_i$ mit. Dann können alle Personen zusammen mit dem chinesischen Restsatz das Geheimnis entschlüsseln. Wählt man die n_i geschickt, so ist es aber keiner Gruppe von $k-1$ Personen möglich mit den ihr zur Verfügung stehenden Informationen das Geheimnis S zu rekonstruieren.

4.3.5 RSA-Kryptosystem

Wir schließen das Kapitel über algebraische Strukturen mit einer wichtigen Anwendung, dem *RSA-Kryptosystem*, ab. Dabei handelt es sich um ein weit verbreitetes asymmetrisches Verschlüsselungsverfahren. Das bedeutet, es wird ein Schlüsselpaar erzeugt, das aus einem öffentlichen Schlüssel besteht, mit dem Nachrichten verschlüsselt werden können, und aus einem privaten Schlüssel, mit dem Nachrichten entschlüsselt werden können. Der Vorteil gegenüber einem symmetrischen Verfahren, bei dem es nur einen Schlüssel gibt, der sowohl für das Verschlüsseln als auch für das Entschlüsseln verwendet wird, liegt auf der Hand. Es ist bei einem asymmetrischen Verfahren nicht notwendig, zunächst auf einem sicheren Weg einen geheimen Schlüssel auszutauschen. Leser, die mit dem Programm OpenPGP vertraut sind, das zum Signieren und Verschlüsseln von E-Mails verwendet werden kann und das unter anderem auf dem RSA-Kryptosystem beruht, werden dies zu schätzen wissen. Das RSA-Kryptosystem ist nach seinen Erfindern Ron Rivest, Adi Shamir und Leonard Adleman benannt, von denen es 1977 vorgestellt wurde.

Schlüsselerzeugung

Als erstes beschäftigen wir uns mit der Schlüsselerzeugung, für deren Beschreibung wir noch eine Definition benötigen.

Definition 4.33. *Die* eulersche Phi-Funktion $\varphi\colon \mathbb{N} \to \mathbb{N}$ *ist für alle* $n \in \mathbb{N}$ *definiert durch*

$$\varphi(n) = |\{k \in \{1, 2, \ldots, n\} \mid \mathrm{ggT}(k, n) = 1\}|.$$

Die Funktion φ *weist somit jeder natürlichen Zahl* n *die Anzahl der zu* n *teilerfremden Zahlen zwischen* 1 *und* n *zu.*

Die eulersche Phi-Funktion hat zahlreiche Anwendungen in der Zahlentheorie. Für zwei verschiedene Primzahlen p und q gilt $\varphi(p) = p-1$, $\varphi(q) = q-1$ und $\varphi(pq) = (p-1)(q-1)$. Dies sollte der Leser sich als Übung überlegen.

Die Schlüsselerzeugung des RSA-Kryptosystems besteht aus drei Schritten.

1. Wähle zwei große Primzahlen p und q mit $p \neq q$ und setze $n = pq$.
2. Wähle ein $e \in \mathbb{N}$ mit $\mathrm{ggT}(e, \varphi(n)) = 1$.

3. Berechne ein $d \in \mathbb{N}$ mit $ed \equiv 1 \bmod \varphi(n)$.

Als öffentlicher Schlüssel wird das Paar (n, e) bekannt gegeben (d. h. es wird zum Beispiel wie bei OpenPGP auf einen öffentlichen Schlüsselserver geladen). Der private Schlüssel, der zum Entschlüsseln und Signieren benutzt wird und geheim gehalten werden muss, ist das Paar (n, d). Alle anderen Zahlen, die in der Schlüsselerzeugung auftreten, werden nicht benötigt und müssen geheim bleiben.

Wir betrachten nun die drei Schritte der Schlüsselerzeugung genauer und überlegen uns, wie sie effizient realisiert werden können. Die Erzeugung großer Primzahlen in Schritt 1 erfolgt in der Regel dadurch, dass eine zufällige ungerade Zahl x in der gewünschten Größenordnung (heutzutage sind das normalerweise Zahlen mit mindestens 1024 Bits) gewählt wird. Dann wird getestet, ob x eine Primzahl ist. Ist x keine Primzahl, so wird getestet, ob $x + 2$ eine Primzahl ist. Ist dies ebenfalls keine Primzahl, so wird $x + 4$ getestet und so weiter. Dieser Prozess wird wiederholt, bis eine Primzahl gefunden wird.

Betrachten wir nun die Laufzeit für das Finden einer Primzahl auf diese Weise. Zum einen stellt sich die Frage, wie viele Zahlen wir testen müssen, bis wir eine Primzahl finden. Ein bekanntes Ergebnis zur Primzahldichte besagt, dass durchschnittlich nur $\ln x$ Zahlen getestet werden müssen, um die erste Primzahl zu finden. Wie lange dauert aber ein einzelner Test, ob eine gegebene Zahl y eine Primzahl ist? Für dieses Problem kennt man bereits seit mehr als 30 Jahren effiziente randomisierte Algorithmen wie zum Beispiel den sogenannten *Miller-Rabin-Test*. Dabei handelt es sich um einen Algorithmus, der zufällige Entscheidungen trifft und mit einer sehr kleinen Wahrscheinlichkeit eine zusammengesetzte Zahl fälschlicherweise als Primzahl deklariert. Diese Fehlerwahrscheinlichkeit ist aber so gering, dass sie in der Praxis keine Rolle spielt. Tatsächlich ist es sehr viel wahrscheinlicher, dass während der Ausführung ein Hardwarefehler auftritt als dass der Algorithmus ein falsches Ergebnis liefert. Insgesamt bedeutet das, dass mithilfe des oben beschriebenen Verfahrens effizient eine große Primzahl gefunden werden kann.

Es sei an dieser Stelle nur kurz erwähnt, dass es jahrzehntelang eine große offene Frage der theoretischen Informatik war, ob es einen effizienten deterministischen Algorithmus gibt, der entscheidet, ob eine gegebene Zahl eine Primzahl ist. Erst 2002 ist es Manindra Agrawal, Neeraj Kayal und Nitin Saxena gelungen, diese Frage positiv zu beantworten, indem sie den ersten effizienten deterministischen Primzahltest angegeben haben. In der Praxis wird dennoch der oben angesprochene Miller-Rabin-Test benutzt, da er deutlich schneller ist und seine Fehlerwahrscheinlichkeit vernachlässigt werden kann.

Schritt 2 kann dadurch realisiert werden, dass solange zufällige Zahlen gewählt werden, bis eine zu $\varphi(n)$ teilerfremde Zahl e gefunden wird. Ob die zufällig erzeugten Zahlen teilerfremd zu $\varphi(n)$ sind, kann effizient mit dem euklidischen Algorithmus bestimmt werden. Aus der Zahlentheorie weiß man, dass die Anzahl der zufälligen Zahlen, die getestet werden müssen, bis eine zu $\varphi(n)$ teilerfremde Zahl gefunden wird, typischerweise höchstens proportional zu $\log \log n$ wächst. Mit hoher Wahrscheinlichkeit genügt es also, wenige Zahlen zu testen.

In Schritt 3 muss das multiplikative Inverse d zu e im Ring $(\mathbb{Z}/\varphi(n)\mathbb{Z}, \oplus_{\varphi(n)}, \odot_{\varphi(n)})$ berechnet werden. Da $\varphi(n) = (p - 1)(q - 1)$ keine Primzahl ist, handelt es sich bei

diesem Ring gemäß Theorem 4.26 nicht um einen Körper. Der Beweis dieses Theorems, den wir in Abschnitt 4.3.2 gegeben haben, liefert eigentlich die folgende allgemeinere Aussage.

Theorem 4.34. *Für $n \in \mathbb{N}$ besitzt ein Element $[\![a]\!]_n$ genau dann ein multiplikatives Inverses in dem Ring $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$, wenn $\mathrm{ggT}(a, n) = 1$ gilt.*

Da wir in Schritt 2 die Zahl e so gewählt haben, dass $\mathrm{ggT}(e, \varphi(n)) = 1$ gilt, garantiert uns dieses Theorem, dass wir in Schritt 3 ein multiplikatives Inverses d finden können. Genauso wie im Beweis von Theorem 4.26 nutzen wir dazu Lemma 4.30, welches das Problem letztendlich mithilfe des euklidischen Algorithmus löst.

Verschlüsselung und Entschlüsselung

Nun haben wir uns davon überzeugt, dass alle drei Schritte der Schlüsselerzeugung bei RSA effizient durchgeführt werden können. Wir können uns nun dem Verschlüsseln und Entschlüsseln von Nachrichten widmen. Wir beschreiben, wie eine Nachricht $m \in \{0, \ldots, n-1\}$ zunächst verschlüsselt und dann wieder entschlüsselt werden kann. Das Objekt, um das es tatsächlich geht (beispielsweise ein längerer Text), muss dann zunächst so in Blöcke zerlegt werden, dass jeder davon als Zahl aus dem Bereich $\{0, \ldots, n-1\}$ codiert werden kann. Ein Text kann beispielsweise in Blöcke von jeweils wenigen Buchstaben zerlegt werden. Ist jeder Buchstabe eines der 256 ASCII-Zeichen, so gibt es $256^k = 2^{8k}$ mögliche Blöcke der Länge k. Die Blocklänge k muss also so gewählt sein, dass $2^{8k} \leq n$ gilt. Um zu verhindern, dass Blöcke mit demselben Inhalt stets auf dieselbe Art codiert werden, wird bei praktischen Implementierungen des RSA-Kryptosystems jeder Block noch um einige zufällige Bits erweitert.

Wir gehen nun davon aus, dass wir bereits eine Zerlegung in Blöcke vorliegen haben, und wir betrachten nur noch das Problem, eine Nachricht $m \in \{0, 1, 2, \ldots, n-1\}$ zu ver- und entschlüsseln.

- **Verschlüsselung:** Bei der Verschlüsselung der Nachricht m wird die Funktion

  $$E\colon \{0, \ldots, n-1\} \to \{0, \ldots, n-1\} \quad \text{mit} \quad E(x) = x^e \bmod n$$

  an der Stelle m ausgewertet. Der Geheimtext der Nachricht $m \in \{0, \ldots, n-1\}$ ist also der Rest von m^e bei Division durch n.

- **Entschlüsselung:** Bei der Entschlüsselung eines Geheimtextes $z \in \{0, \ldots, n-1\}$ wird die Funktion

  $$D\colon \{0, \ldots, n-1\} \to \{0, \ldots, n-1\} \quad \text{mit} \quad D(x) = x^d \bmod n$$

  an der Stelle z ausgewertet. Der Klartext zum Geheimtext $z \in \{0, 1, 2, \ldots, n-1\}$ ist also der Rest von z^d bei Division durch n.

Als erstes überlegen wir uns, dass die Entschlüsselung wirklich invers zu der Verschlüsselung ist.

Theorem 4.35. *Für jedes $m \in \{0, \dots, n-1\}$ gilt $D(E(m)) = m$.*

Zum Beweis dieses Theorems benötigen wir den folgenden *Satz von Euler*.

Theorem 4.36. *Für jedes $n \in \mathbb{N}$ und jedes $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$ gilt die Kongruenz $a^{\varphi(n)} \equiv 1 \bmod n$.*

Beweis. Wir erinnern uns daran, dass $(\mathbb{Z}/n\mathbb{Z})^*$ die Menge der Einheiten aus $\mathbb{Z}/n\mathbb{Z}$ bezeichnet, also die Elemente, die ein multiplikatives Inverses besitzen. Gemäß Theorem 4.34 gilt

$$(\mathbb{Z}/n\mathbb{Z})^* = \{[\![x]\!] \mid x \in \{0, \dots, n-1\} \text{ und } \mathrm{ggT}(x, n) = 1\}$$

und gemäß der Definition der eulerschen Phi-Funktion gilt $\varphi(n) = |(\mathbb{Z}/n\mathbb{Z})^*|$.

Für ein $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$ betrachten wir die Abbildung $f\colon (\mathbb{Z}/n\mathbb{Z})^* \to (\mathbb{Z}/n\mathbb{Z})^*$ mit $f([\![x]\!]) = [\![ax]\!]$ für alle $x \in \{0, \dots, n-1\}$. Diese Abbildung ist injektiv, denn seien $x, y \in \{0, \dots, n-1\}$ mit $f([\![x]\!]) = f([\![y]\!])$ gegeben, so gilt

$$\begin{aligned}
[\![x]\!] &= [\![a]\!]^{-1} \odot_n ([\![a]\!] \odot_n [\![x]\!]) \\
&= [\![a]\!]^{-1} \odot_n f([\![x]\!]) \\
&= [\![a]\!]^{-1} \odot_n f([\![y]\!]) \\
&= [\![a]\!]^{-1} \odot_n ([\![a]\!] \odot_n [\![y]\!]) \\
&= [\![y]\!].
\end{aligned}$$

In dieser Rechnung haben wir ausgenutzt, dass es wegen $\mathrm{ggT}(a, n) = 1$ ein inverses Element $[\![a]\!]^{-1}$ zu $[\![a]\!]$ in $(\mathbb{Z}/n\mathbb{Z}, \oplus_n, \odot_n)$ gibt. Genau wie im Beweis des chinesischen Restsatzes, folgt aus der Injektivität die Bijektivität von f, da die Definitionsmenge und die Zielmenge dieselbe endliche Kardinalität besitzen.

Sei $(\mathbb{Z}/n\mathbb{Z})^* = \{[\![r_1]\!], \dots, [\![r_{\varphi(n)}]\!]\}$ für Zahlen $r_i \in \{0, \dots, n-1\}$. Da die Funktion f bijektiv ist und somit nur die Reihenfolge der Elemente aus $(\mathbb{Z}/n\mathbb{Z})^*$ vertauscht, gilt

$$\begin{aligned}
[\![r_1]\!] \odot_n \dots \odot [\![r_{\varphi(n)}]\!] &= [\![ar_1]\!] \odot_n \dots \odot [\![ar_{\varphi(n)}]\!] \\
&= [\![r_1]\!] \odot_n \dots \odot [\![r_{\varphi(n)}]\!] \odot_n [\![a^{\varphi(n)}]\!].
\end{aligned}$$

Da es zu jedem $[\![r_i]\!]$ ein Inverses $[\![r_i]\!]^{-1}$ gibt, können wir diese Gleichung gemäß der Kürzungsregel zu

$$[\![1]\!] = [\![a^{\varphi(n)}]\!]$$

umformen. Das bedeutet, $a^{\varphi(n)}$ lässt bei Division durch n den Rest 1. □

Nun können wir die Korrektheit des RSA-Kryptosystems nachweisen.

Beweis von Theorem 4.35. Wir weisen mithilfe einer Fallunterscheidung nach, dass für jedes $m \in \{0, \dots, n-1\}$ die Gleichung $[\![(m^e)^d]\!]_n = [\![m]\!]_n$ gilt.

- Wir betrachten zunächst den wahrscheinlichen Fall, dass $\mathrm{ggT}(m,n) = 1$ gilt. Wegen $ed \equiv 1 \bmod \varphi(n)$ gibt es ein $h \in \mathbb{Z}$ mit $ed = 1 + h\varphi(n)$. Es gilt

$$m^{ed} = m^{1+h\varphi(n)} = m(m^{\varphi(n)})^h$$

und somit

$$[\![m^{ed}]\!]_n = [\![m]\!]_n \odot_n ([\![m^{\varphi(n)}]\!]_n)^h = [\![m]\!]_n \odot_n ([\![1]\!]_n)^h = [\![m]\!]_n,$$

wobei wir im vorletzten Schritt den Satz von Euler (Theorem 4.36) angewendet haben.

- Nun betrachten wir den Fall, dass m von genau einer der Primzahlen p und q geteilt wird. Es gelte ohne Beschränkung der Allgemeinheit $p \mid m$ und nicht $q \mid m$. Dann gilt $m^{ed} \equiv m \equiv 0 \bmod p$. Außerdem gilt $\mathrm{ggT}(m,q) = 1$ und somit können wir analog zum ersten Fall argumentieren, dass $m^{ed} \equiv m \bmod q$ gilt:

$$\begin{aligned}[\![m^{ed}]\!]_q &= [\![m]\!]_q \odot_q ([\![m^{\varphi(n)}]\!]_q)^h = [\![m]\!]_q \odot_q ([\![m^{\varphi(p)\varphi(q)}]\!]_q)^h \\ &= [\![m]\!]_q \odot_q ([\![m^{\varphi(q)}]\!]_q)^{\varphi(p)h} = [\![m]\!]_q \odot_q ([\![1]\!]_q)^{\varphi(p)h} = [\![m]\!]_q.\end{aligned}$$

Wegen der Teilerfremdheit von p und q implizieren diese beiden Kongruenzen $m^{ed} \equiv m \bmod (pq)$ (der Leser erinnere sich an den Beweis vom chinesischen Restsatz, wo wir ein ähnliches Argument verwendet haben). Wegen $n = pq$ ist $[\![m^{ed}]\!]_n = [\![m]\!]_n$ damit gezeigt.

- Falls $m \in \{0, \ldots, n-1\}$ von beiden Primzahlen p und q geteilt wird, so gilt $m = 0$. In diesem Fall gilt $[\![m^{ed}]\!]_n = [\![m]\!]_n = [\![0]\!]_n$.

Damit ist der Beweis der Korrektheit des RSA-Kryptosystems abgeschlossen. □

Effiziente Implementierung von Verschlüsselung und Entschlüsselung

Es bleibt die Frage zu klären, wie die Verschlüsselung und die Entschlüsselung effizient implementiert werden können. Da die Exponenten e und d sehr groß werden können (Zahlen in der Größenordnung von 2^{1024} oder sogar 2^{4096} sind heutzutage keine Seltenheit bei der Anwendung von RSA), kommt es nicht in Frage $x^e \bmod n$ und $x^d \bmod n$ mithilfe von e bzw. d Multiplikationen und einer anschließenden Division mit Rest zu berechnen. Man benötigt ein effizienteres Verfahren, dass als *binäre Exponentiation* oder *schnelles Potenzieren* bekannt ist. Die Anzahl der Rechenoperationen, die dieses Verfahren benötigt, wächst nur proportional zum Logarithmus des Exponenten.

Die binäre Exponentiation ist besonders schnell, wenn in der Binärdarstellung des Exponenten viele Nullen enthalten sind. Die Verschlüsselung kann also beschleunigt werden, wenn ein Exponent e mit möglichst vielen Nullen gewählt wird. Auf die Anzahl der Nullen im Exponenten d haben wir keinen direkten Einfluss, da d sich als multiplikatives Inverses zu e ergibt. Wir können die Entschlüsselung aber mithilfe des chinesischen Restsatzes beschleunigen. Erweitern wir den privaten Schlüssel

um die Zahlen p und q, so können wir beim Entschlüsseln eines Geheimtextes z zunächst die Zahlen $m_p = z^d \bmod p$ und $m_q = z^d \bmod q$ berechnen. Die gesuchte Nachricht $m \in \{0, 1, \ldots, n\}$ ist dann die eindeutige Lösung des Kongruenzsystems

$$m \equiv m_p \bmod p$$
$$m \equiv m_q \bmod q.$$

Diese kann mit dem chinesischen Restsatz berechnet werden. Dazu werden, wie im konstruktiven Beweis beschrieben, $y_p \in \mathbb{Z}$ und $y_q \in \mathbb{Z}$ mit $y_p p + y_q q = 1$ berechnet. Die gesuchte Lösung m ergibt sich dann als $m = (m_p y_q q + m_q y_p p) \bmod n$. Die Zahlen y_p und y_q können einmal vorberechnet werden. Der Vorteil dieses Verfahrens ist, dass die binäre Exponentiation in $\mathbb{Z}/p\mathbb{Z}$ und $\mathbb{Z}/q\mathbb{Z}$ und nicht in $\mathbb{Z}/n\mathbb{Z}$ durchgeführt werden muss. Da es sich hierbei um deutlich kleinere Zahlen handelt, spart dieses Vorgehen Zeit.

Sicherheit des RSA-Kryptosystems

Zum Schluss wollen wir noch kurz die Sicherheit des RSA-Kryptosystems thematisieren. Im Prinzip kann der private Schlüssel (n, d) aus dem öffentlichen Schlüssel (n, e) berechnet werden. Dazu genügt es, die Zahl n in ihre Primfaktoren p und q zu zerlegen. Ist dies gelungen, so ist insbesondere $\varphi(n)$ bekannt und der Exponent d kann, genau wie in der Schlüsselerzeugung beschrieben, effizient aus dem Exponenten e berechnet werden.

Trotz intensiver Forschung ist es allerdings bis heute nicht gelungen, einen effizienten Algorithmus zur Faktorisierung von großen Zahlen zu finden. Als effizient würde man in diesem Falle einen Algorithmus ansehen, dessen Laufzeit zum Faktorisieren einer Zahl n durch ein Polynom in $\ln n$ nach oben abgeschätzt werden kann. Ein Algorithmus, der maximal $10 \cdot \ln^3 n + 7$ Schritte benötigt, wäre also beispielsweise effizient. Der schnellste bekannte Algorithmus, das sogenannte *Zahlkörpersieb*, zum Faktorisieren hat eine Laufzeit, die für eine geeignete Konstante $C \approx 1{,}9$ proportional zu $e^{C(\ln n)^{1/3}(\ln \ln n)^{2/3}}$ wächst. Dieser Algorithmus ist für große Zahlen mit 1024 oder mehr Bits so langsam, dass damit kein realistischer Angriff auf das RSA-Kryptosystem möglich ist. Man vermutet sogar, dass es keinen effizienten Algorithmus zur Faktorisierung gibt. Diese Vermutung ist aber bis heute unbewiesen.

Es stellt sich die Frage, ob die Berechnung der Primfaktoren p und q aus dem öffentlichen Schlüssel durch Faktorisierung von n der einzige mögliche Angriff auf das RSA-Kryptosystem ist. A priori kann es ja durchaus sein, dass der private Schlüssel (n, d) auch auf andere Weise aus dem öffentlichen Schlüssel (n, e) berechnet werden kann oder dass eine effiziente Entschlüsselung sogar ohne Kenntnis des privaten Schlüssels möglich ist. Auch dies ist bis heute unklar, aber man vermutet natürlich, dass es keinen Algorithmus gibt, der nur basierend auf dem öffentlichen Schlüssel eine effiziente Entschlüsselung durchführt. Der Leser mag aus dieser Diskussion also mitnehmen, dass niemand weiß, ob das weit verbreitete RSA-Kryptosystem wirklich sicher ist, und dass die vermeintliche Sicherheit im Moment darauf beruht, dass es bislang trotz vieler Versuche noch nicht gelungen ist, RSA erfolgreich anzugreifen.

Beispiel

Wir betrachten nun noch ein Beispiel für das RSA-Kryptosystem. Wir wählen $p = 3$ und $q = 11$. Dann ist $n = 33$ und $\varphi(n) = 20$. Wir können Nachrichten $m \in \{0, \ldots, 32\}$ ver- und entschlüsseln. Wir machen es uns einfach und codieren längere Texte Buchstabe für Buchstabe. Dabei steht $m = 0$ für ein Leerzeichen, $m = 1$ für den Buchstaben A, $m = 2$ für den Buchstaben B und so weiter. Wählen wir $e = 7$, so gilt, wie gewünscht, $\mathrm{ggT}(e, \varphi(n)) = 1$. Ferner gilt $(-1) \cdot \varphi(n) + 3 \cdot e = 1$. Wir setzen dementsprechend $d = 3$ und erhalten $ed = 21 \equiv 1 \bmod \varphi(n)$. Damit ist die Schlüsselerzeugung abgeschlossen. Der öffentliche Schlüssel ist $(33, 7)$ und der private Schlüssel ist $(33, 3)$.

Möchten wir nun das Wort „RSA“ verschlüsseln, so fangen wir mit dem ersten Buchstaben R an. Dieser entspricht der Nachricht $m = 18$. Diese Nachricht wird zu $18^7 \bmod 33$ verschlüsselt, was 6 ergibt. Führen wir dieses Verfahren auch für die Buchstaben S und A durch, so erhalten wir den Geheimtext „06 13 01“.

22 15 14 12 00 14 06 30 27 12 28 00 29 14 15 00 16 14 20 00 01 20 13 26 14 02 14 20 16 14 20 00 25 06 21 14 30 21 20 28 14 20

Kapitel 5

Einführung in die mathematische Logik

Wir werden in diesem Kapitel eine kleine Einführung in die *mathematische Logik* geben. Dabei handelt es sich um ein Teilgebiet der Mathematik und Informatik, das sich damit beschäftigt, wie man formal Schlüsse zieht und Beweise führt. Auf den ersten Blick mag dieses Thema sehr theoretisch wirken, es hat aber zahlreiche praktische Anwendungen in der Informatik. Mathematische Logik wird beispielsweise im Bereich der künstlichen Intelligenz eingesetzt, um Wissen so zu repräsentieren, dass daraus sinnvolle Schlüsse gezogen werden können. Außerdem bildet Logik die Grundlage von Datenbanksprachen wie SQL und sie spielt bei der Verifikation von Hardware und Software eine große Rolle.

Wir werden uns in diesem Kapitel mit *Aussagenlogik* und *Prädikatenlogik* beschäftigen. In der Aussagenlogik untersucht man Ausdrücke, die durch einfache Verknüpfungen aus *atomaren Aussagen* entstehen. Jede solche atomare Aussage kann entweder wahr oder falsch sein und man interessiert sich dafür, wie der Wahrheitswert eines Ausdrucks von den Wahrheitswerten der atomaren Aussagen abhängt. Aus mathematischer Sicht ist die Aussagenlogik nicht besonders interessant, da sie relativ ausdrucksschwach ist (d. h. viele interessante Sachverhalte können mithilfe der vorhandenen Verknüpfungen nicht ausgedrückt werden). Zum einen ist die Aussagenlogik aber die Grundlage für kompliziertere Logiken und eignet sich deshalb gut als Einstieg und zur Illustration wichtiger Konzepte, und zum anderen treten bereits im Kontext der Aussagenlogik interessante algorithmische Probleme auf, die von grundlegender Bedeutung für die Informatik sind. Von besonderem Interesse ist das *Erfüllbarkeitsproblem*, also die Frage, ob es für einen gegebenen Ausdruck Wahrheitswerte für die atomaren Aussagen gibt, für die er wahr ist.

Eine Erweiterung der Aussagenlogik ist die *Prädikatenlogik*. In dieser Erweiterung sind zusätzlich zu den vorhandenen Verknüpfungen der Aussagenlogik die Quantoren $\forall$ und $\exists$ erlaubt, die wir in den vergangenen Kapiteln schon eingeführt und benutzt haben. Die Prädikatenlogik ist deutlich ausdrucksstärker und wir können viele interessante Sachverhalte als prädikatenlogische Formeln darstellen.

Die Inhalte dieses Kapitels stammen aus dem Buch von Uwe Schöning [8] sowie den Skripten von Erich Grädel [3] und Nicole Schweikardt [7].

5.1 Aussagenlogik

Bereits in Abschnitt 2.2 haben wir uns mit Aussagenlogik beschäftigt, ohne diesen Begriff jedoch explizit zu benutzen. Wir haben dort bereits Beispiele für Aussagen betrachtet und verschiedene Verknüpfungen definiert, die es erlauben, mehrere Aussagen miteinander zu einer neuen Aussage zu kombinieren. In diesem Abschnitt führen wir die Aussagenlogik systematischer ein und diskutieren einige Aspekte, die bereits auf die Betrachtung der Prädikatenlogik vorbereiten.

5.1.1 Syntax

Zunächst definieren wir, was wir unter einer aussagenlogischen Formel verstehen. Dazu gehen wir davon aus, dass eine abzählbar unendliche Menge $\mathrm{AV} = \{x_1, x_2, x_3, \ldots\}$ von *Aussagenvariablen* gegeben ist. Bei diesen Aussagenvariablen handelt es sich um die atomaren Aussagen, aus denen alle anderen Ausdrücke zusammengesetzt sind.

Definition 5.1. *Die Menge* AL *der* aussagenlogischen Formeln *ist die kleinste Sprache über dem Alphabet* $\mathrm{AV} \cup \{\mathbf{0}, \mathbf{1}, \wedge, \vee, \neg, \rightarrow, \leftrightarrow, (,)\}$ *mit den folgenden drei Eigenschaften.*

a) Es gilt $\mathbf{0} \in \mathrm{AL}$ *und* $\mathbf{1} \in \mathrm{AL}$*.*

b) Für jede Aussagenvariable $x \in \mathrm{AV}$ *gilt* $x \in \mathrm{AL}$*.*

c) Sind $\varphi_1 \in \mathrm{AL}$ *und* $\varphi_2 \in \mathrm{AL}$ *zwei aussagenlogische Formeln, so sind auch die Wörter* $\neg\varphi_1$*,* $(\varphi_1 \wedge \varphi_2)$*,* $(\varphi_1 \vee \varphi_2)$*,* $(\varphi_1 \rightarrow \varphi_2)$ *und* $(\varphi_1 \leftrightarrow \varphi_2)$ *aussagenlogische Formeln aus* AL*.*

Aussagenlogische Formeln aus AL sind zum Beispiel

$$(x_1 \wedge x_2),\ (\neg x_1 \rightarrow x_3),\ \neg\neg x_2,\ \neg(x_3 \vee x_2) \text{ und } (x_2 \leftrightarrow (x_1 \vee x_2)),$$

nicht aber

$$x_1 \wedge x_2,\ ((\neg x_1 \vee x_3),\ \neg(x_2) \text{ und } x_1 \vee x_2 \vee x_3.$$

Wie wir bereits in Abschnitt 2.2 besprochen haben, nennen wir die Verknüpfungen $\wedge$, $\vee$ und $\neg$ auch *Konjunktion*, *Disjunktion* und *Negation*. Außerdem nennen wir die Verknüpfungen $\rightarrow$ und $\leftrightarrow$ *Implikation* und *Äquivalenz*. Wir weisen an dieser Stelle aber ausdrücklich darauf hin, dass wir bisher in diesem Kapitel nur die *Syntax* der Aussagenlogik definiert haben. Das heißt, wir wissen, welche Zeichenfolgen gültige Formeln sind und welche nicht. Wir haben jedoch noch nicht die *Semantik*, also die Bedeutung, von aussagenlogischen Formeln diskutiert.

Um die Lesbarkeit von Formel zu verbessern und Klammern zu sparen, vereinbaren wir, dass $\neg$ vor $\wedge$ und $\vee$ ausgewertet wird, die wiederum vor $\rightarrow$ und $\leftrightarrow$ ausgewertet werden. Außerdem dürfen die äußeren Klammern eines Ausdrucks weggelassen werden. Mit diesen Konventionen ist der Ausdruck $x_1 \wedge \neg x_2 \rightarrow x_3$ beispielsweise eine Abkürzung für den Ausdruck $((x_1 \wedge \neg x_2) \rightarrow x_3)$. Auch lassen wir wieder Konjunktionen und

Disjunktionen von mehr als zwei Formeln ohne Klammern zu. Dabei gehen wir implizit davon aus, dass eine Klammerung von links nach rechts erfolgt. Der Ausdruck $x_1 \wedge x_2 \wedge x_3 \wedge x_4$ ist also beispielsweise eine Abkürzung für $(((x_1 \wedge x_2) \wedge x_3) \wedge x_4)$.

Bevor wir auf die Semantik der Aussagenlogik eingehen, möchten wir den Leser an das Konzept der strukturellen Induktion erinnern, das wir im Kontext von regulären Ausdrücken in Lemma 3.21 kennengelernt haben. Dieses Beweisprinzip können wir aufgrund der induktiven Definition auch auf aussagenlogische Formeln anwenden. Möchten wir zeigen, dass jede aussagenlogische Formel $\varphi \in \mathrm{AL}$ eine gewisse Eigenschaft E erfüllt, so genügt es Folgendes zu zeigen.

- Induktionsanfang: Die Formeln $\mathbf{0}$, $\mathbf{1}$ und jede Formel $x \in \mathrm{AV}$ erfüllen Eigenschaft E.
- Induktionsschritt: Sind $\varphi_1, \varphi_2 \in \mathrm{AL}$ zwei Formeln, die Eigenschaft E erfüllen, so erfüllen auch die Formeln $\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \to \varphi_2)$ und $(\varphi_1 \leftrightarrow \varphi_2)$ Eigenschaft E.

Mithilfe der strukturellen Induktion (in diesem Kontext auch *Induktion über den Formelaufbau* genannt) ist es möglich, nachzuweisen, dass kein echtes Präfix (Anfangsstück) einer aussagenlogischen Formel selbst eine aussagenlogische Formel ist (bezogen auf Definition 5.1 und nicht auf die eingeführten Abkürzungen). Beispielsweise ist kein Präfix von $((x_1 \vee x_2) \wedge x_3)$ eine gültige aussagenlogische Formel aus AL. Damit folgt eine einfache, aber ganz wesentliche Eigenschaft von aussagenlogischen Formeln, nämlich ihre *eindeutige Lesbarkeit*. Das bedeutet, dass für jede Formel eindeutig feststeht, wie sie in ihre unmittelbaren Bestandteile zerlegt werden kann. Gilt für $\varphi \in \mathrm{AL}$ beispielsweise $\varphi = (\varphi_1 \circ \varphi_2)$ für $\circ \in \{\wedge, \vee, \to, \leftrightarrow\}$ und $\varphi_1, \varphi_2 \in \mathrm{AL}$ und gleichzeitig $\varphi = (\varphi_1' \circ' \varphi_2')$ für $\circ' \in \{\wedge, \vee, \to, \leftrightarrow\}$ und $\varphi_1', \varphi_2' \in \mathrm{AL}$, so ist $\circ = \circ'$, $\varphi_1 = \varphi_1'$ und $\varphi_2 = \varphi_2'$.

Die eindeutige Lesbarkeit hat zur Folge, dass auch induktive Definitionen über den Formelaufbau eindeutig sind. Beispielsweise können wir die *Tiefe* $d(\varphi)$ *einer Formel* $\varphi \in \mathrm{AL}$ wie folgt induktiv definieren.

- Es sei $d(\mathbf{0}) = d(\mathbf{1}) = 0$. Außerdem sei für alle $x \in \mathrm{AV}$ ebenfalls $d(x) = 0$.
- Für eine Formel $\varphi \in \mathrm{AL}$ sei $d(\neg\varphi) = d(\varphi) + 1$.
- Für eine Verknüpfung $\circ \in \{\wedge, \vee, \to, \leftrightarrow\}$ und Formeln $\varphi_1, \varphi_2 \in \mathrm{AL}$ sei $d((\varphi_1 \circ \varphi_2)) = \max\{d(\varphi_1), d(\varphi_2)\} + 1$.

Ebenfalls können wir für eine Formel $\varphi \in \mathrm{AL}$ die Menge $\mathrm{Var}(\varphi)$ der in φ vorkommenden Variablen induktiv wie folgt definieren.

- Es sei $\mathrm{Var}(\mathbf{0}) = \mathrm{Var}(\mathbf{1}) = \emptyset$. Außerdem sei $\mathrm{Var}(x) = \{x\}$ für alle $x \in \mathrm{AV}$.
- Für eine Formel $\varphi \in \mathrm{AL}$ sei $\mathrm{Var}(\neg\varphi) = \mathrm{Var}(\varphi)$.
- Für eine Verknüpfung $\circ \in \{\wedge, \vee, \to, \leftrightarrow\}$ und Formeln $\varphi_1, \varphi_2 \in \mathrm{AL}$ sei $\mathrm{Var}((\varphi_1 \circ \varphi_2)) = \mathrm{Var}(\varphi_1) \cup \mathrm{Var}(\varphi_2)$.

Es gilt beispielsweise

$$\begin{aligned}\mathrm{Var}((x_2 \wedge (x_1 \vee \neg x_2))) &= \mathrm{Var}(x_2) \cup \mathrm{Var}((x_1 \vee \neg x_2)) = \{x_2\} \cup (\mathrm{Var}(x_1) \cup \mathrm{Var}(\neg x_2)) \\ &= \{x_2\} \cup (\{x_1\} \cup \mathrm{Var}(x_2)) = \{x_2\} \cup (\{x_1\} \cup \{x_2\}) = \{x_1, x_2\}.\end{aligned}$$

5.1.2 Semantik

Auch die Semantik der Aussagenlogik können wir induktiv definieren. Bevor wir dies tun, benötigen wir aber noch eine weitere Definition.

Definition 5.2. *Eine* Bewertung *oder* Interpretation *ist eine Abbildung* $B\colon X \to \{0,1\}$ *für ein* $X \subseteq \mathrm{AV}$. *Die Bewertung* $B\colon X \to \{0,1\}$ *heißt* passend *zu einer Formel* $\varphi \in \mathrm{AL}$, *wenn* $\mathrm{Var}(\varphi) \subseteq X$ *gilt.*

Eine Bewertung weist also einigen Aussagenvariablen Wahrheitswerte zu. Ist $\varphi' \in \mathrm{AL}$ eine Teilformel von $\varphi \in \mathrm{AL}$ (das heißt ein Teil der Formel φ, der für sich selbst genommen ebenfalls eine gültige Formel ist), so gilt offensichtlich $\mathrm{Var}(\varphi') \subseteq \mathrm{Var}(\varphi)$. Das bedeutet, eine Bewertung B, die zu einer Formel φ passt, passt auch zu jeder Teilformel von φ. Diese Eigenschaft nutzen wir in der folgenden Definition implizit aus.

Definition 5.3. *Ist* $B\colon X \to \{0,1\}$ *eine zu* $\varphi \in \mathrm{AL}$ *passende Bewertung, so besitzt die Formel* φ *einen eindeutigen Wahrheitswert* $[\![\varphi]\!]_B \in \{0,1\}$. *Dieser ist induktiv wie folgt definiert.*

- *Es sei* $[\![\mathbf{0}]\!]_B = 0$, $[\![\mathbf{1}]\!]_B = 1$ *und* $[\![x]\!]_B = B(x)$ *für alle* $x \in X$.
- *Für eine Formel* $\varphi \in \mathrm{AL}$, *zu der die Bewertung* B *passt, sei* $[\![\neg\varphi]\!]_B = 1 - [\![\varphi]\!]_B$.
- *Es seien* $\varphi_1, \varphi_2 \in \mathrm{AL}$ *zwei Formeln, zu denen die Bewertung* B *passt. Dann sei*

$$\begin{aligned}[\![\varphi_1 \wedge \varphi_2]\!]_B &= \min\{[\![\varphi_1]\!]_B, [\![\varphi_2]\!]_B\}, \\ [\![\varphi_1 \vee \varphi_2]\!]_B &= \max\{[\![\varphi_1]\!]_B, [\![\varphi_2]\!]_B\}, \\ [\![\varphi_1 \to \varphi_2]\!]_B &= [\![(\neg\varphi_1 \vee \varphi_2)]\!]_B, \\ [\![\varphi_1 \leftrightarrow \varphi_2]\!]_B &= \begin{cases} 1 & \textit{falls } [\![\varphi_1]\!]_B = [\![\varphi_2]\!]_B, \\ 0 & \textit{sonst.} \end{cases}\end{aligned}$$

Gilt $[\![\varphi]\!]_B = 1$, *so sagen wir, dass die Formel* φ *für die Bewertung* B wahr *ist. Gilt hingegen* $[\![\varphi]\!]_B = 0$, *so sagen wir, dass die Formel* φ *für die Bewertung* B falsch *ist.*

Genauso wie in Abschnitt 2.2 hätten wir in Definition 5.3 auch Wahrheitstabellen benutzen können. Der Leser sollte sich davon überzeugen, dass die Formeln, die wir in der Definition benutzt haben, den Wahrheitstabellen aus Abschnitt 2.2 entsprechen, wobei $\to$ als Implikation und $\leftrightarrow$ als Äquivalenz zu interpretieren ist.

Wir betrachten als Beispiel die Formel $\varphi = \neg x_1 \vee (x_2 \wedge \neg x_3)$ und die dazu passende Bewertung B mit $B(x_1) = 1$, $B(x_2) = 1$ und $B(x_3) = 0$. Es gilt

$$\begin{aligned}
[\![\varphi]\!]_B &= \max\{[\![\neg x_1]\!]_B, [\![(x_2 \wedge \neg x_3)]\!]_B\} \\
&= \max\{1 - [\![x_1]\!]_B, \min\{[\![x_2]\!]_B, [\![\neg x_3]\!]_B\}\} \\
&= \max\{1 - B(x_1), \min\{B(x_2), 1 - [\![x_3]\!]_B\}\} \\
&= \max\{0, \min\{1, 1 - B(x_3)\}\} \\
&= \max\{0, \min\{1, 1\}\} \\
&= \max\{0, 1\} = 1.
\end{aligned}$$

Für eine gegebene aussagenlogische Formel ist es oft wichtig, zu entscheiden, ob es eine Bewertung gibt, für die sie wahr ist, oder ob sie gar für jede Bewertung wahr ist.

Definition 5.4. *Es sei $\varphi \in$ AL eine aussagenlogische Formel und $\Phi \subseteq$ AL eine Menge von Formeln.*

a) Eine zu φ passende Bewertung B heißt Modell von φ*, wenn $[\![\varphi]\!]_B = 1$ gilt. Wir sagen dann, dass die Bewertung B die Formel φ* erfüllt *und schreiben $B \models \varphi$. Wir nennen B ein* Modell der Formelmenge Φ*, wenn B alle Formeln aus Φ erfüllt (d. h. insbesondere, dass B zu allen Formeln aus Φ passt) und schreiben dann $B \models \Phi$.*

b) Die Formel φ heißt erfüllbar*, wenn es ein Modell für sie gibt, d. h. eine zu ihr passende Bewertung B mit $[\![\varphi]\!]_B = 1$.*

c) Die Formel φ heißt gültig*, wenn $[\![\varphi]\!]_B = 1$ für jede zu ihr passende Bewertung B gilt. In diesem Falle wird die Formel φ auch als* Tautologie *bezeichnet und wir schreiben $\models \varphi$.*

Beispiele

- Die Formel x_1 ist erfüllbar (jede Bewertung B mit $B(x_1) = 1$ erfüllt sie), aber nicht gültig (jede Bewertung B mit $B(x_1) = 0$ erfüllt sie nicht).
- Die Formel $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$ ist erfüllbar, aber nicht gültig. Eine erfüllende Bewertung ist $B(x_1) = B(x_2) = B(x_3) = 0$. Eine nicht erfüllende Bewertung ist $B(x_1) = B(x_2) = B(x_3) = 1$.
- Die Formel $(x_1 \vee x_2) \vee \neg x_1$ ist erfüllbar und gültig.
- Die Formel $(x_1 \wedge x_2) \wedge (\neg x_1 \wedge x_2)$ ist nicht erfüllbar und nicht gültig.

Eine einleuchtende Aussage, die man formal mithilfe von struktureller Induktion nachweisen kann, ist das folgende *Koinzidenzlemma*.

Lemma 5.5. *Es sei $\varphi \in$ AL eine Formel und es seien B und B' zwei zu φ passende Bewertungen, sodass $B(x) = B'(x)$ für alle $x \in \mathrm{Var}(\varphi)$ gilt. Dann gilt $[\![\varphi]\!]_B = [\![\varphi]\!]_{B'}$.*

Das folgende Lemma setzt die Begriffe erfüllbar und gültig zueinander in Beziehung.

Lemma 5.6. *Eine Formel φ ist genau dann erfüllbar, wenn $\neg\varphi$ keine Tautologie ist.*

Beweis. Ist die Formel φ erfüllbar, so gibt es eine zu ihr passende Bewertung B mit $[\![\varphi]\!]_B = 1$. Für diese Bewertung gilt $[\![\neg\varphi]\!]_B = 1 - [\![\varphi]\!]_B = 0$. Somit ist $\neg\varphi$ keine Tautologie.

Ist die Formel $\neg\varphi$ keine Tautologie, so gibt es eine zu ihr passende Bewertung B mit $[\![\neg\varphi]\!]_B = 0$. Für diese Bewertung gilt $[\![\varphi]\!]_B = 1 - [\![\neg\varphi]\!]_B = 1$. Somit ist φ erfüllbar. □

Beispiel

Bevor wir mit den theoretischen Überlegungen fortfahren, möchten wir demonstrieren, dass mithilfe von aussagenlogischen Formeln bereits interessante Probleme modelliert werden können. Die meisten Leser sind vermutlich mit Sudoku-Rätseln vertraut. Bei einem solchen Rätsel ist ein 9×9-Gitter gegeben, das in neun 3×3-Blöcke unterteilt ist. Jede Zelle des Gitters ist mit einer Zahl zwischen 1 und 9 zu füllen, sodass jede Zahl in jeder Zeile, in jeder Spalte und in jedem Block genau einmal vorkommt. Einige Zellen sind dabei bereits mit Zahlen gefüllt. Die folgende Abbildung zeigt ein Beispiel für ein typisches Sudoku-Rätsel.

4			1	9			2	
					4			9
1	9			7		8		
9	7	5		4	6			
2	4	6				3	5	7
			2	5		4	9	6
		9		1			7	3
8			6					
	3			2	5			1

Wir möchten nun eine aussagenlogische Formel aufstellen, die genau dann erfüllbar ist, wenn das Sudoku-Rätsel lösbar ist. Außerdem soll jedes Modell der Formel eine Lösung des Rätsels liefern. Der erste Schritt bei einer solchen Modellierungsaufgabe besteht darin, geeignete Variablen zu identifizieren. Statt der vorgegebenen Variablennamen $x_1, x_2, x_3, \ldots$ benutzt man häufig intuitivere Namen. Eine naheliegende Idee bei Sudoku ist es, für jede Zelle des Gitters eine Variable einzuführen, die angibt, welche Zahl sie enthält. Das Problem mit dieser Idee ist jedoch, dass wir ausschließlich binäre Variablen zur Verfügung haben, die entweder wahr oder falsch sein können. Deshalb führen wir für jede Zelle $(i, j) \in G = \{1, 2, \ldots, 9\}^2$ des Gitters und jede mögliche Zahl $k \in \{1, 2, \ldots, 9\}$ eine Variable $x_{i,j}^k$ ein. Diese Variable ist so zu interpretieren, dass sie genau dann den Wert 1 annimmt, wenn in der Zelle (i, j) die Zahl k steht.

Im nächsten Schritt der Modellierung müssen die Variablen sinnvoll zueinander in Beziehung gesetzt werden. Als erstes möchten wir sicherstellen, dass in jeder Zelle genau eine Zahl steht. Dazu konstruieren wir zunächst eine Formel φ_1, die sicherstellt, dass in jeder Zelle mindestens eine Zahl steht:

$$\varphi_1 = \bigwedge_{(i,j)\in G} \left(\bigvee_{k=1}^{9} x_{i,j}^k \right).$$

Dabei sind die Verknüpfungen $\bigwedge$ und $\bigvee$ ähnlich zu lesen wie ein Summenzeichen $\sum$, d. h. $\bigvee_{k=1}^{9} x_{i,j}^k$ steht beispielsweise für die Disjunktion der Variablen $x_{i,j}^1, \ldots, x_{i,j}^9$. Für eine gegebene Zelle $(i,j) \in G$ besagt der Ausdruck $\bigvee_{k=1}^{9} x_{i,j}^k$ also, dass in der Zelle (i,j) mindestens eine Zahl steht. Die Formel φ_1 ist eine Konjunktion dieser Ausdrücke über alle Zellen des Gitters. Damit besagt die Formel φ_1, dass in jeder Zelle des Gitters mindestens eine Zahl steht. Als nächstes möchten wir eine Formel angeben, die codiert, dass in jeder Zelle des Gitters höchstens eine Zahl steht. Diese Formel nennen wir φ_2 und wir können sie schreiben als

$$\varphi_2 = \bigwedge_{(i,j)\in G} \left(\bigwedge_{k=1}^{9} \left(\bigwedge_{\ell \in \{1,\ldots,9\}\setminus\{k\}} \neg(x_{i,j}^k \wedge x_{i,j}^\ell) \right) \right).$$

Diese Formel besagt, dass in keiner Zelle (i,j) des Gitters zwei verschiedene Zahlen k und ℓ stehen. Damit haben wir erreicht, dass in jedem Modell der Formel $\varphi_1 \wedge \varphi_2$ in jeder Zelle genau eine Zahl steht.

Nun bleibt noch zu codieren, dass in keiner Zeile, in keiner Spalte und in keinem Block eine Zahl doppelt vorkommen darf. Da alle Zeilen, Spalten und Blöcke aus jeweils genau neun Zellen bestehen, ist diese Aussage äquivalent dazu, dass jede Zahl zwischen 1 und 9 in jeder Zeile, in jeder Spalte und in jedem Block mindestens einmal vorkommt. Diese drei Aussagen codieren wir wie folgt:

$$\varphi_3 = \bigwedge_{j=1}^{9} \left(\bigwedge_{k=1}^{9} \left(\bigvee_{i=1}^{9} x_{i,j}^k \right) \right), \qquad \text{(Zeilen)}$$

$$\varphi_4 = \bigwedge_{i=1}^{9} \left(\bigwedge_{k=1}^{9} \left(\bigvee_{j=1}^{9} x_{i,j}^k \right) \right), \qquad \text{(Spalten)}$$

$$\varphi_5 = \bigwedge_{i,j\in\{0,1,2\}} \left(\bigwedge_{k=1}^{9} \left(\bigvee_{i',j'\in\{1,2,3\}} x_{3i+i',3j+j'}^k \right) \right). \qquad \text{(Blöcke)}$$

Wir haben somit erreicht, dass in jedem Modell der Formel

$$\varphi_{\text{Spielregeln}} = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5$$

in jeder Zelle genau eine Zahl steht und dass in jeder Zeile, in jeder Spalte und in jedem Block jede Zahl zwischen 1 und 9 genau einmal vorkommt. Es bleibt nun noch die gegebenen Anfangswerte zu codieren. Dies ist einfach, wir müssen dazu nur die entsprechenden Variablen auf 1 setzen. In dem obigen Beispiel müssten wir beispielsweise erreichen, dass die Variablen $x_{1,1}^4, x_{4,1}^1, x_{5,1}^9, x_{8,1}^2, \ldots, x_{9,9}^1$ in jedem Modell den Wert 1 haben. Dies erreichen wir einfach dadurch, dass wir eine Konjunktion über diese Variablen bilden. Diese nennen wir φ_{Anfang}. In obigem Beispiel gilt also

$$\varphi_{\text{Anfang}} = x_{1,1}^4 \wedge x_{4,1}^1 \wedge x_{5,1}^9 \wedge x_{8,1}^2 \wedge \ldots \wedge x_{9,9}^1.$$

Insgesamt codiert die Formel $\varphi = \varphi_{\text{Spielregeln}} \wedge \varphi_{\text{Anfang}}$ das gegebene Sudoku-Rätsel. Diese Formel ist genau dann erfüllbar, wenn das Rätsel eine Lösung besitzt. Aus jedem Modell für φ, d. h. aus jeder Bewertung der Variablen $x_{i,j}^k$, die die Formel φ erfüllt, kann direkt eine Lösung des Rätsels abgelesen werden.

Es kommt oft vor, dass zwei unterschiedliche Formeln, dieselbe Bedeutung haben. Dies werden wir im Rest dieses Abschnittes genauer betrachten.

Definition 5.7. *Zwei Formeln $\varphi_1 \in \mathrm{AL}$ und $\varphi_2 \in \mathrm{AL}$ heißen* (logisch) äquivalent, *wenn $[\![\varphi_1]\!]_B = [\![\varphi_2]\!]_B$ für jede Bewertung gilt, die zu beiden Formeln passt. Wir schreiben dann $\varphi_1 \equiv \varphi_2$.*

Der Leser überlege sich als Übung, dass $\equiv$ eine Äquivalenzrelation auf der Menge AL ist. Außerdem sollte er sich klarmachen, was der Unterschied zwischen $\varphi_1 \leftrightarrow \varphi_2$ und $\varphi_1 \equiv \varphi_2$ ist. Den Beweis des folgenden Theorems überlassen wir ebenfalls dem Leser als Übung. Er folgt mit einfachen Rechnungen aus Definition 5.3.

Theorem 5.8. *Es seien $\varphi_1, \varphi_2, \varphi_3 \in \mathrm{AL}$ beliebige aussagenlogische Formeln. Dann gelten die folgenden logischen Äquivalenzen.*

$$(\varphi_1 \wedge \varphi_1) \equiv \varphi_1 \qquad \textit{(Idempotenz)}$$
$$(\varphi_1 \vee \varphi_1) \equiv \varphi_1$$
$$(\varphi_1 \wedge \varphi_2) \equiv (\varphi_2 \wedge \varphi_1) \qquad \textit{(Kommutativität)}$$
$$(\varphi_1 \vee \varphi_2) \equiv (\varphi_2 \vee \varphi_1)$$
$$((\varphi_1 \wedge \varphi_2) \wedge \varphi_3) \equiv (\varphi_1 \wedge (\varphi_2 \wedge \varphi_3)) \qquad \textit{(Assoziativität)}$$
$$((\varphi_1 \vee \varphi_2) \vee \varphi_3) \equiv (\varphi_1 \vee (\varphi_2 \vee \varphi_3))$$
$$\neg\neg\varphi_1 \equiv \varphi_1 \qquad \textit{(Elimination der doppelten Negation)}$$
$$\neg(\varphi_1 \wedge \varphi_2) \equiv (\neg\varphi_1 \vee \neg\varphi_2) \qquad \textit{(De Morgan'sche Gesetze)}$$
$$\neg(\varphi_1 \vee \varphi_2) \equiv (\neg\varphi_1 \wedge \neg\varphi_2)$$
$$(\varphi_1 \wedge (\varphi_2 \vee \varphi_3)) \equiv ((\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)) \qquad \textit{(Distributivgesetze)}$$
$$(\varphi_1 \vee (\varphi_2 \wedge \varphi_3)) \equiv ((\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3))$$
$$(\varphi_1 \wedge (\varphi_1 \vee \varphi_2)) \equiv \varphi_1 \qquad \textit{(Absorption)}$$
$$(\varphi_1 \vee (\varphi_1 \wedge \varphi_2)) \equiv \varphi_1$$
$$(\varphi_1 \to \varphi_2) \equiv (\neg\varphi_2 \to \neg\varphi_1) \qquad \textit{(Kontraposition)}$$
$$(\varphi_1 \to \varphi_2) \equiv (\neg\varphi_1 \vee \varphi_2) \qquad \textit{(Elimination der Implikation)}$$
$$(\varphi_1 \leftrightarrow \varphi_2) \equiv ((\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)) \qquad \textit{(Elimination der Äquivalenz)}$$

5.1.3 Normalformen

Die Syntax der Aussagenlogik, die wir in Definition 5.1 kennengelernt haben, erlaubt es, die vorhandenen Verknüpfungen beliebig zu kombinieren. Diese Flexibilität ist bei der Modellierung von Problemen durch aussagenlogische Formeln zwar hilfreich, möchte man aber beispielsweise einen Algorithmus entwerfen, der testet, ob eine gegebene Formel erfüllbar ist, so ist es praktisch, wenn man nur Formeln mit einer gewissen Struktur betrachten muss. Aus Theorem 5.8 folgt beispielsweise, dass man sich auf Formeln beschränken kann, die die Verknüpfungen $\to$ und $\leftrightarrow$ nicht enthalten, da diese Verknüpfungen durch äquivalente Formeln beschrieben werden können, die nur die Verknüpfungen $\neg$, $\wedge$ und $\vee$ enthalten.

Wir werden in diesem Abschnitt noch einen Schritt weitergehen und verschiedene *Normalformen* von aussagenlogischen Formeln kennenlernen. Eine Normalform ist eine eingeschränkte Klasse von Formeln aus AL, sodass es zu jeder beliebigen aussagenlogische Formel eine äquivalente Formel in der entsprechenden Normalform gibt. Zunächst definieren wir die *konjunktive Normalform* und die *disjunktive Normalform*. Dabei handelt es sich um sehr wichtige Normalformen, die in der Informatik eine große Rolle spielen und denen der Leser im Laufe des Informatikstudiums noch oft begegnen wird.

Definition 5.9. *Eine Formel der Form x oder $\neg x$ für $x \in \mathrm{AV}$ nennen wir ein* Literal. *Ein Literal der Form x nennen wir* positives Literal *und ein Literal der Form $\neg x$ nennen wir* negatives Literal.

a) Eine aussagenlogische Formel $\varphi \in \mathrm{AL}$ ist in konjunktiver Normalform (KNF), *wenn sie eine Konjunktion von Disjunktionen von Literalen ist, d. h. wenn sie die Gestalt*

$$\varphi = \bigwedge_{i=1}^{n} \Big(\bigvee_{j=1}^{m_i} \ell_{i,j} \Big)$$

hat, wobei $n, m_1, \ldots, m_n \in \mathbb{N}$ gilt und $\ell_{i,j}$ für jedes i und j ein Literal ist.

Die Teilformeln $\bigvee_{j=1}^{m_i} \ell_{i,j}$ nennen wir die Klauseln *von φ.*

b) Eine aussagenlogische Formel $\varphi \in \mathrm{AL}$ ist in disjunktiver Normalform (DNF), *wenn sie eine Disjunktion von Konjunktionen von Literalen ist, d. h. wenn sie die Gestalt*

$$\varphi = \bigvee_{i=1}^{n} \Big(\bigwedge_{j=1}^{m_i} \ell_{i,j} \Big)$$

hat, wobei $n, m_1, \ldots, m_n \in \mathbb{N}$ gilt und $\ell_{i,j}$ für jedes i und j ein Literal ist.

Theorem 5.10. *Zu jeder aussagenlogischen Formel $\varphi \in \mathrm{AL}$ gibt es äquivalente aussagenlogische Formeln in konjunktiver und disjunktiver Normalform.*

Beweis. Sei φ eine beliebige aussagenlogische Formel und sei $n = |\mathrm{Var}(\varphi)|$ die Anzahl verschiedener Variablen in φ. Es gelte ohne Beschränkung der Allgemeinheit $\mathrm{Var}(\varphi) = \{x_1, \ldots, x_n\}$. Zunächst erstellen wir eine Wahrheitstabelle für die Formel φ. In dieser Tabelle tragen wir für jede der 2^n möglichen Bewertungen der Variablen den Wert von φ ein. Für eine Bewertung der Variablen können wir mithilfe von Definition 5.3 leicht den Wert von φ ermitteln.

Anhand dieser Wahrheitstabelle können wir direkt eine zu φ äquivalente Formel in disjunktiver Normalform bestimmen. Steht in allen Zeilen der Wahrheitstabelle als Ergebnis 0, so ist die Formel φ nicht erfüllbar und die ebenfalls nicht erfüllbare Formel $(x_1 \wedge \neg x_1)$ in disjunktiver Normalform ist äquivalent zu φ. Wir betrachten nun den Fall, dass es mindestens eine erfüllende Bewertung gibt. In diesem Fall bezeichnen wir mit $\mathcal{B}$ die Menge aller erfüllenden Bewertungen, die genau den Variablen aus $\mathrm{Var}(\varphi)$ Werte zuweisen. Für jede Bewertung $B \in \mathcal{B}$ definieren wir eine Formel $\varphi_B = \ell_1^B \wedge \cdots \wedge \ell_n^B$ mit

$$\ell_i^B = \begin{cases} \neg x_i & \text{falls } B(x_i) = 0, \\ x_i & \text{falls } B(x_i) = 1. \end{cases}$$

Man überzeugt sich leicht davon, dass die Bewertung B die einzige Bewertung ist, die die Formel φ_B erfüllt. Das bedeutet, eine Bewertung B erfüllt die Disjunktion der Formeln $\varphi_{B'}$ für $B' \in \mathcal{B}$ genau dann, wenn $B \in \mathcal{B}$ gilt, wenn also B eine erfüllende Bewertung von φ ist. Wir können die gesuchte zu φ äquivalente Formel in disjunktiver Normalform demnach wie folgt wählen:

$$\varphi_{\mathrm{DNF}} = \bigvee_{B \in \mathcal{B}} \varphi_B.$$

Auch für die gesuchte Formel in konjunktiver Normalform hilft uns die Wahrheitstabelle von φ. Wir erzeugen zunächst mit dem oben beschriebenen Verfahren eine Formel φ' in disjunktiver Normalform, die äquivalent zu der Formel $\neg\varphi$ ist. Es sei

$$\varphi' = \bigvee_{i=1}^{k} \Big(\bigwedge_{j=1}^{m_i} \ell_{i,j} \Big)$$

mit $n, m_1, \ldots, m_k \in \mathbb{N}$, wobei $\ell_{i,j}$ für jedes i und j ein Literal ist. Mit den De Morgan'schen Gesetzen aus Theorem 5.8 erhalten wir

$$\varphi \equiv \neg\varphi' \equiv \neg \bigvee_{i=1}^{k} \Big(\bigwedge_{j=1}^{m_i} \ell_{i,j} \Big) \equiv \bigwedge_{i=1}^{k} \Big(\neg \bigwedge_{j=1}^{m_i} \ell_{i,j} \Big) \equiv \bigwedge_{i=1}^{k} \Big(\bigvee_{j=1}^{m_i} \neg\ell_{i,j} \Big).$$

Die letzte Formel ist äquivalent zu φ und in konjunktiver Normalform, wenn alle vorkommenden doppelten Negationen gestrichen werden. □

Beispiel

Wir betrachten die Formel

$$\varphi = (((x_1 \wedge \neg x_2) \vee \neg(\neg x_1 \vee x_3)) \wedge (\neg x_3 \vee x_2)) \vee (\neg x_1 \wedge x_2 \wedge x_3)$$

und stellen zunächst die Wahrheitstabelle von φ auf.

x_1	x_2	x_3	φ
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	0

Es gibt drei Bewertungen der Variablen, für die die Formel φ wahr ist. Entsprechend Theorem 5.10 erzeugen wir zu jeder dieser Bewertungen eine Konjunktion von Literalen. Anschließend bilden wir die Disjunktion dieser Konjunktionen, um die folgende zu φ äquivalente Formel in DNF zu erhalten:

$$(x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3).$$

Um eine zu φ äquivalente Formel in KNF zu erhalten, erstellen wir zunächst eine zu $\neg\varphi$ äquivalente Formel φ' in DNF. Wir verwenden dazu dieselbe Wahrheitstabelle wie oben und erstellen nun für jede Bewertung, für die die Formel φ falsch ist, eine Konjunktion von Literalen. Dies ergibt die Formel

$$\varphi' = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3).$$

Um nun aus φ' eine zu φ äquivalente Formel in KNF zu erhalten, wenden wir die De Morgan'schen Gesetze an. Dies ergibt die Formel

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

Ein Nachteil des Verfahrens aus Theorem 5.10 zur Umwandlung einer beliebigen Formel in äquivalente Formeln in konjunktiver und disjunktiver Normalform ist, dass es stets notwendig ist, die gesamte Wahrheitstabelle zu erstellen. Dies ist bei Formeln mit vielen Variablen nicht effizient möglich, da die Größe der Wahrheitstabelle exponentiell mit der Anzahl der Variablen wächst. In vielen Fällen kann man Formeln effizienter in konjunktive und disjunkte Normalform umwandeln, indem man die Regeln aus Theorem 5.8 geschickt anwendet. Basierend auf diesen Regeln geben wir ein Verfahren an, um beliebige Formeln in äquivalente Formeln in konjunktiver Normalform umzuwandeln. Dieses Verfahren ist für viele Formeln effizienter als das aus dem Beweis von Theorem 5.10.

ErzeugeKNF(φ)

1. Solange es in φ für beliebige $\varphi_1, \varphi_2 \in \mathrm{AL}$ eine Teilformel der Form $(\varphi_1 \leftrightarrow \varphi_2)$ gibt, ersetze diese durch die Formel $((\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1))$.
2. Solange es in φ für beliebige $\varphi_1, \varphi_2 \in \mathrm{AL}$ eine Teilformel der Form $(\varphi_1 \rightarrow \varphi_2)$ gibt, ersetze diese durch die Formel $(\neg\varphi_1 \vee \varphi_2)$.
3. Solange es in φ für beliebige $\varphi_1, \varphi_2 \in \mathrm{AL}$ eine Teilformel der Form $\neg\neg\varphi_1$, $\neg(\varphi_1 \wedge \varphi_2)$ oder $\neg(\varphi_1 \vee \varphi_2)$ gibt, nimm eine der folgenden Ersetzungen vor.

   Ersetze $\neg\neg\varphi_1$ durch φ_1.
   Ersetze $\neg(\varphi_1 \wedge \varphi_2)$ durch $(\neg\varphi_1 \vee \neg\varphi_2)$.
   Ersetze $\neg(\varphi_1 \vee \varphi_2)$ durch $(\neg\varphi_1 \wedge \neg\varphi_2)$.

4. Solange es in φ für beliebige $\varphi_1, \varphi_2, \varphi_3 \in \mathrm{AL}$ eine Teilformel der Form $(\varphi_1 \vee (\varphi_2 \wedge \varphi_3))$ oder $((\varphi_1 \wedge \varphi_2) \vee \varphi_3)$ gibt, nimm eine der folgenden Ersetzungen vor.

   Ersetze $(\varphi_1 \vee (\varphi_2 \wedge \varphi_3))$ durch $((\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3))$.
   Ersetze $((\varphi_1 \wedge \varphi_2) \vee \varphi_3)$ durch $((\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3))$.

Theorem 5.11. *Der Algorithmus ErzeugeKNF(φ) erzeugt zu jeder Formel $\varphi \in \mathrm{AL}$ in endlich vielen Schritten eine äquivalente Formel in konjunktiver Normalform.*

Wir werden dieses Theorem nicht formal beweisen, sondern nur die Beweisidee grob skizzieren. Es muss gezeigt werden, dass die Formel, die der Algorithmus erzeugt, äquivalent zu der Formel φ ist, dass der Algorithmus nach endlich vielen Schritten terminiert und dass die Formel, die der Algorithmus erzeugt, in konjunktiver Normalform ist.

Zum Beweis der ersten Aussage genügt es zu zeigen, dass die aktuelle Formel während der Ausführung des Algorithmus zu jedem Zeitpunkt äquivalent zu der Formel φ ist, die der Algorithmus als Eingabe erhält. Formal kann man dies mithilfe von vollständiger Induktion über die Anzahl an bereits erfolgten Ersetzungen nachweisen. Für den Induktionsanfang beobachten wir, dass die aktuelle Formel vor der ersten Ersetzung gleich φ und somit äquivalent zu φ ist. Für den Induktionsschritt gehen wir davon aus, dass die aktuelle Formel φ' durch n Ersetzungen aus φ entstanden und äquivalent zu φ ist. Sei nun φ'' eine Formel, die durch eine mögliche Ersetzung in einem der vier Schritte aus φ' entsteht. Wir müssen zeigen, dass φ'' äquivalent zu φ' und somit auch zu φ ist. Dies folgt aus den Tautologien aus Theorem 5.8, denn in jedem Schritt wird eine Teilformel durch eine äquivalente Teilformel ersetzt. Die Schritte 1 und 2 entsprechen der Elimination von Äquivalenzen und Implikationen. In Schritt 3 werden die De Morgan'sche Gesetze und die Elimination der doppelten Negation angewendet. Schritt 4 benutzt eines der Distributivgesetze und die Kommutativität der Disjunktion.

Der Beweis, dass der Algorithmus stets nach endlich vielen Schritten terminiert, ist etwas schwieriger. Die Schritte 1 und 2 sind unproblematisch, da sie jeweils eine Äquivalenz bzw. eine Implikation aus der Formel entfernen. Dies ist nur so oft möglich, wie es Äquivalenzen bzw. Implikationen gibt. Schritt 3 kann ebenfalls zu keiner Endlosschleife führen. Dazu überlegt man sich, dass die Anwendung der De Morgan'schen Gesetze dazu führt, dass die Negationen „nach innen" wandern. Nach endlich vielen Schritten stehen alle noch vorhandenen Negationen direkt vor Variablen. Dann ist Schritt 3 beendet. Ähnlich kann man in Schritt 4 argumentieren, dass die Anwendung der Distributivgesetze dazu führt, dass die Disjunktionen „nach innen" wandern. Auch dies kann nur endlich oft passieren. Auf eine Formalisierung dieses Argumentes und einen Beweis, dass die erzeugte Formel wirklich in konjunktiver Normalform ist, verzichten wir aus Zeitgründen.

Beispiel

Zum Abschluss führen wir den Algorithmus ErzeugeKNF(φ) noch exemplarisch auf der Formel

$$\varphi = (\neg(x_2 \to x_1) \vee (x_1 \wedge x_3))$$

aus. Der Algorithmus erzeugt die folgende Sequenz von zu φ äquivalenten Formeln.

$$\begin{aligned}
\varphi &= (\neg(x_2 \to x_1) \vee (x_1 \wedge x_3)) \\
&\equiv (\neg(\neg x_2 \vee x_1) \vee (x_1 \wedge x_3)) && \text{(Schritt 2)} \\
&\equiv ((\neg\neg x_2 \wedge \neg x_1) \vee (x_1 \wedge x_3)) && \text{(Schritt 3)} \\
&\equiv ((x_2 \wedge \neg x_1) \vee (x_1 \wedge x_3))
\end{aligned}$$

$$\equiv ((x_2 \vee (x_1 \wedge x_3)) \wedge (\neg x_1 \vee (x_1 \wedge x_3))) \qquad \text{(Schritt 4)}$$
$$\equiv (((x_2 \vee x_1) \wedge (x_2 \vee x_3)) \wedge (\neg x_1 \vee (x_1 \wedge x_3)))$$
$$\equiv (((x_2 \vee x_1) \wedge (x_2 \vee x_3)) \wedge ((\neg x_1 \vee x_1) \wedge (\neg x_1 \vee x_3)))$$

Auf die letzte Formel lässt sich keine Ersetzungsregel aus Schritt 4 mehr anwenden. Diese Formel ist wie gewünscht in konjunktiver Normalform. Mit den eingeführten Abkürzungen können wir sie auch wie folgt schreiben:

$$(x_2 \vee x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_1) \wedge (\neg x_1 \vee x_3).$$

5.1.4 Resolutionskalkül

Unter einem *Kalkül* versteht man in der Logik eine Menge von syntaktischen Umformungsregeln, mit denen aus gegebenen Formeln neue Formeln erzeugt werden können. Das Ziel bei dem Entwurf eines Kalküls ist es, die Umformungsregeln so zu wählen, dass sie ein algorithmisches Verfahren ergeben, mit dem getestet werden kann, ob eine gegebene Formel erfüllbar ist.

Der Sinn von Kalkülen erschließt sich in der Aussagenlogik nicht sofort, denn die Erfüllbarkeit einer gegebenen aussagenlogischen Formel kann leicht algorithmisch überprüft werden, indem für alle möglichen Bewertungen der in der Formel enthaltenen Variablen getestet wird, ob sie die Formel erfüllen. In komplizierteren Logiken wie der Prädikatenlogik gibt es jedoch keine so einfache Möglichkeit mehr, die Erfüllbarkeit einer gegebenen Formel zu testen, und man muss stattdessen auf Kalküle zurückgreifen. Die Betrachtung von Kalkülen in der Aussagenlogik kann also zum einen als Vorbereitung auf die Prädikatenlogik gesehen werden und zum anderen sind die Algorithmen, die aus Kalkülen hervorgehen, für viele Formeln effizienter als der oben beschriebene einfache Erfüllbarkeitstest.

Bevor wir die theoretische Betrachtung von Kalkülen fortsetzen, geben wir zunächst mit dem *Resolutionskalkül* einen konkreten Kalkül für die Aussagenlogik an. Der Resolutionskalkül testet, ob eine gegebene Formel $\varphi \in \mathrm{AL}$ in konjunktiver Normalform erfüllbar ist oder nicht. Formeln, die nicht in konjunktiver Normalform vorliegen, müssen vor der Anwendung des Resolutionskalküls zunächst in konjunktive Normalform gebracht werden. Dazu kann eines der Verfahren aus Abschnitt 5.1.3 genutzt werden. Der Resolutionskalkül operiert auf den Klauseln der Formel und fügt der aktuellen Formel in jedem Schritt neue Klauseln hinzu.

Zur einfachen Beschreibung dieses Kalküls ist es nützlich, einige Bezeichnungen einzuführen. Sei

$$\varphi = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} \ell_{i,j} \right)$$

eine beliebige aussagenlogische Formel in konjunktiver Normalform. Wir stellen jede Klausel als die Vereinigung der in ihr vorkommenden Literale dar. Die Formel φ besteht

dementsprechend aus den Klauseln $C_1, \ldots, C_n$ mit

$$C_i = \{\ell_{i,j} \mid j \in \{1, \ldots, m_i\}\}.$$

Mit $\mathcal{K}(\varphi) = \{C_1, \ldots, C_n\}$ bezeichnen wir die Menge der Klauseln der Formel φ. Aus der Menge $\mathcal{K}(\varphi)$ kann man die Formel φ nicht eindeutig rekonstruieren, da für $\mathcal{K}(\varphi)$ die Reihenfolge der Klauseln und die Reihenfolge der Literale in den Klauseln keine Rolle spielt. Außerdem sind Literale, die mehrfach in derselben Klausel vorkommen, und Klauseln, die mehrfach in φ vorkommen, in $\mathcal{K}(\varphi)$ nur einmal enthalten. Beispielsweise haben die drei Formeln

$$\begin{aligned}
\varphi_1 &= (\neg x_1 \vee x_2) \wedge x_3, \\
\varphi_2 &= (x_3 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_2) \text{ und} \\
\varphi_3 &= (x_2 \vee \neg x_1) \wedge x_3 \wedge (\neg x_1 \vee x_2)
\end{aligned}$$

dieselbe Klauselmenge

$$\mathcal{K}(\varphi_1) = \mathcal{K}(\varphi_2) = \mathcal{K}(\varphi_3) = \{\{\neg x_1, x_2\}, \{x_3\}\}.$$

Für die Erfüllbarkeit spielt dies aber keine Rolle, denn alle Formeln mit derselben Klauselmenge sind äquivalent, was der Leser als Übung begründen sollte.

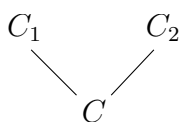
Um Begriffe wie erfüllbar und äquivalent von Formeln auf Klauselmengen zu übertragen, identifizieren wir im Folgenden eine Klauselmenge K mit der Formel

$$\varphi_K = \bigwedge_{C \in K} \Big(\bigvee_{\ell \in C} \ell \Big)$$

mit $\mathcal{K}(\varphi_K) = K$. Für eine Bewertung B, der in K vorkommenden Variablen, setzen wir $[\![K]\!]_B = [\![\varphi_K]\!]_B$ und $[\![C]\!]_B = [\![\bigvee_{\ell \in C} \ell]\!]_B$ für alle $C \in K$. Eine Klauselmenge K ist dementsprechend genau dann erfüllbar, wenn es eine Bewertung der in K vorkommenden Variablen gibt, für die in jeder Klausel $C \in K$ mindestens ein Literal wahr ist. Man beachte, dass die leere Klauselmenge gemäß dieser Definition erfüllbar ist. Mit $\square$ bezeichnen wir die leere Klausel, die kein Literal enthält. Eine Klauselmenge, die die Klausel $\square$ enthält, ist gemäß obiger Definition nicht erfüllbar. Zwei Klauselmengen sind äquivalent, wenn ihre Wahrheitswerte für jede Bewertung der Variablen übereinstimmen.

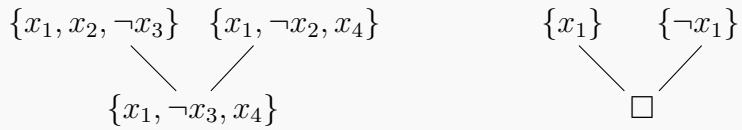
Der Resolutionskalkül basiert auf der folgenden einfachen Umformungsregel.

Definition 5.12. *Es seien C_1 und C_2 Klauseln und es sei $x \in \mathrm{AV}$ eine Variable mit $x \in C_1$ und $\neg x \in C_2$. Die* Resolvente *der Klauseln C_1 und C_2 ist die Klausel $C = (C_1 \setminus \{x\}) \cup (C_2 \setminus \{\neg x\})$. Wir stellen dies graphisch wie folgt dar.*

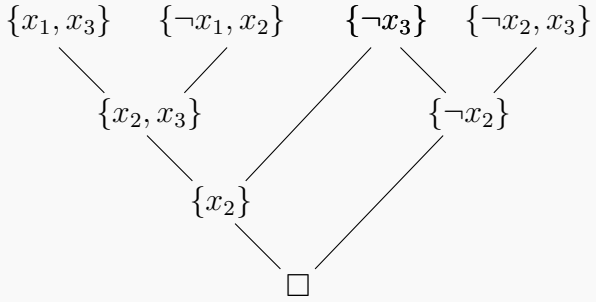


Beispiele

Die folgenden Abbildungen zeigen zwei Beispiele für Resolutionen.



Die folgende Abbildung zeigt ein Beispiel, in dem aus vier gegebenen Klauseln durch mehrere Resolutionen die Klausel □ abgeleitet wird.



Das folgende Lemma ist die Grundlage des Resolutionskalküls.

Lemma 5.13. *Es sei K eine Klauselmenge und es seien $C_1 \in K$ und $C_2 \in K$ zwei Klauseln, sodass es eine Variable $x \in \mathrm{AV}$ mit $x \in C_1$ und $\neg x \in C_2$ gibt. Ist C die Resolvente der beiden Formeln C_1 und C_2, dann sind die Klauselmengen K und $K \cup \{C\}$ äquivalent.*

Beweis. In den Klauselmengen K und $K \cup \{C\}$ kommen genau dieselben Variablen vor. Sei B eine Bewertung dieser Variablen mit $[\![K \cup \{C\}]\!]_B = 1$. Dann gilt insbesondere $[\![K]\!]_B = 1$.

Sei nun B eine Bewertung mit $[\![K]\!]_B = 1$. Für diese Bewertung gilt $[\![C_1]\!]_B = [\![C_2]\!]_B = 1$. Wir unterscheiden zwei Fälle.

1. Ist $B(x) = 0$, so ist $[\![C_1 \setminus \{x\}]\!]_B = 1$, da sonst $[\![C_1]\!]_B = 0$ wäre. Es gibt also in $C_1 \setminus \{x\}$ mindestens ein erfülltes Literal, welches per Definition auch in C enthalten ist. Damit gilt $[\![C]\!]_B = 1$ und insgesamt $[\![K \cup \{C\}]\!]_B = 1$.

2. Ist $B(x) = 1$, so ist $[\![C_2 \setminus \{\neg x\}]\!]_B = 1$, da sonst $[\![C_2]\!]_B = 0$ wäre. Es gibt also in $C_2 \setminus \{\neg x\}$ mindestens ein erfülltes Literal, welches per Definition auch in C enthalten ist. Damit gilt $[\![C]\!]_B = 1$ und insgesamt $[\![K \cup \{C\}]\!]_B = 1$.

Damit ist gezeigt, dass die Klauselmenge K genau dann erfüllbar ist, wenn die Klauselmenge $K \cup \{C\}$ erfüllbar ist. □

Im Resolutionskalkül werden der aktuellen Klauselmenge solange Resolventen hinzugefügt, wie es noch Resolventen gibt, die noch nicht in der Klauselmenge enthalten sind. Formal werden dabei die Mengen aus der folgenden Definition erzeugt.

Definition 5.14. *Für eine Klauselmenge K sei*

$$\mathrm{Res}(K) = K \cup \{C \mid C \text{ ist Resolvente zweier Klauseln aus } K\}.$$

Es sei außerdem

$$\mathrm{Res}^0(K) = K \quad \text{und} \quad \mathrm{Res}^{n+1}(K) = \mathrm{Res}(\mathrm{Res}^n(K)) \text{ für } n \in \mathbb{N}_0$$

und

$$\mathrm{Res}^*(K) = \bigcup_{n \in \mathbb{N}_0} \mathrm{Res}^n(K).$$

Aus der Definition folgt direkt

$$K = \mathrm{Res}^0(K) \subseteq \mathrm{Res}^1(K) \subseteq \mathrm{Res}^2(K) \subseteq \mathrm{Res}^3(K) \subseteq \cdots \subseteq \mathrm{Res}^*(K).$$

Beispiel

Sei $K = \{\{x_1, x_3\}, \{\neg x_1, x_2\}, \{\neg x_3\}\}$. Dann gilt

$$\mathrm{Res}(K) = K \cup \{\{x_2, x_3\}, \{x_1\}\},$$
$$\mathrm{Res}^2(K) = \mathrm{Res}(K) \cup \{\{x_2\}\}$$

und

$$\mathrm{Res}^2(K) = \mathrm{Res}^3(K) = \mathrm{Res}^4(K) = \ldots = \mathrm{Res}^*(K).$$

Das folgende Lemma besagt, dass man die Menge $\mathrm{Res}^*(K)$ aus Definition 5.1.4 für jede endliche Klauselmenge in endlich vielen Schritten berechnen kann.

Lemma 5.15. *Für jede endlich Klauselmenge K gibt es ein $n \in \mathbb{N}$ mit*

$$\mathrm{Res}^n(K) = \mathrm{Res}^{n+1}(K) = \mathrm{Res}^{n+2}(K) = \ldots = \mathrm{Res}^*(K).$$

Beweis. Es genügt zu zeigen, dass es ein $n \in \mathbb{N}$ mit $\mathrm{Res}^n(K) = \mathrm{Res}^{n+1}(K)$ gibt, denn aus dieser Gleichung folgt, dass $\mathrm{Res}^m(K) = \mathrm{Res}^n(K)$ für alle $m > n$ und somit auch $\mathrm{Res}^*(K) = \mathrm{Res}^n(K)$ gilt. Für $m = n + 2$ folgt dies beispielsweise wegen

$$\mathrm{Res}^{n+2}(K) = \mathrm{Res}(\mathrm{Res}^{n+1}(K)) = \mathrm{Res}(\mathrm{Res}^n(K)) = \mathrm{Res}^{n+1}(K) = \mathrm{Res}^n(K).$$

Dieses Argument kann man mittels vollständiger Induktion auf alle $m > n$ übertragen.

Zum Beweis des Lemmas führen wir einen Widerspruchsbeweis und gehen davon aus, dass es kein $n \in \mathbb{N}$ mit $\mathrm{Res}^n(K) = \mathrm{Res}^{n+1}(K)$ gibt. Da $\mathrm{Res}^n(K) \subseteq \mathrm{Res}^{n+1}(K)$ für alle $n \in \mathbb{N}$ gilt, folgt aus der Annahme $|\mathrm{Res}^{n+1}(K)| \geq |\mathrm{Res}^n(K)| + 1$ für alle $n \in \mathbb{N}$. Wegen $|K| \geq 0$ gilt somit $|\mathrm{Res}^n(K)| \geq n$ für jedes $n \in \mathbb{N}$.

Es sei m die Anzahl an verschiedenen Variablen, die in der Klauselmenge K vorkommen. Da bei der Resolution keine neuen Variablen eingefügt werden, kommen auch in den Klauselmengen $\mathrm{Res}^n(K)$ nur diese Variablen vor. Da es insgesamt nur $2m$ verschiedene Literale gibt, die in den Klauseln vorkommen können, gibt es nur 2^{2m} mögliche Klauseln. Da in Klauselmengen per Definition keine Klauseln mehrfach enthalten sein können, gilt $|\mathrm{Res}^n(K)| \leq 2^{2m}$ für jedes $n \in \mathbb{N}$. Dies ist aber ein Widerspruch zu der Aussage, dass $|\mathrm{Res}^n(K)| \geq n$ für jedes $n \in \mathbb{N}$ gilt. Damit ist das Lemma bewiesen. □

Der Resolutionskalkül beruht auf dem folgenden Theorem.

Theorem 5.16. *Eine endliche Klauselmenge K ist genau dann unerfüllbar, wenn $\square \in \mathrm{Res}^*(K)$ gilt.*

Beweis. Die eine Richtung des Beweises ist mit dem bisher Gesagten einfach. Gemäß Lemma 5.15 existiert ein $m \in \mathbb{N}$ mit $\mathrm{Res}^*(K) = \mathrm{Res}^m(K)$. Da das Hinzufügen einer Resolvente gemäß Lemma 5.13 stets zu einer äquivalenten Klauselmenge führt, ist $\mathrm{Res}^m(K)$ genau dann erfüllbar, wenn K erfüllbar ist. Ist $\square \in \mathrm{Res}^m(K)$, so sind $\mathrm{Res}^m(K) = \mathrm{Res}^*(K)$ und damit auch K unerfüllbar.

Es bleibt zu zeigen, dass $\square \in \mathrm{Res}^m(K)$ gilt, wenn K nicht erfüllbar ist. Dies weisen wir mit vollständiger Induktion über die Anzahl n an verschiedenen Variablen, die in K vorkommen, nach. Ohne Beschränkung der Allgemeinheit seien dies die Variablen $x_1, \ldots, x_n$. Genau diese Variablen kommen auch in jeder Menge $\mathrm{Res}^m(K)$ für $m \in \mathbb{N}$ vor. Sei im Folgenden K eine nicht erfüllbare Klauselmenge.

Für den Induktionsanfang betrachten wir den Fall $n = 0$. In diesem Fall kommen in der Klauselmenge K gar keine Variablen vor. Somit ist die leere Klausel $\square$ die einzige Klausel, die in K enthalten sein kann. Dies lässt als Möglichkeiten für die Klauselmenge nur $K = \emptyset$ und $K = \{\square\}$ zu. Die leere Klauselmenge ist erfüllbar. Von Interesse für diese Richtung des Beweises ist also nur die Klauselmenge $K = \{\square\}$. Diese ist unerfüllbar und es gilt wie gewünscht $\square \in \mathrm{Res}^*(K)$, da $K \subseteq \mathrm{Res}^*(K)$.

Für den Induktionsschritt sei $n \in \mathbb{N}$ und es sei K eine beliebige unerfüllbare Klauselmenge, in der genau die Variablen $x_1, \ldots, x_n$ vorkommen. Wir konstruieren zunächst die beiden Klauselmengen

$$K^+ = \{C \setminus \{\neg x_n\} \mid C \in K, x_n \notin C\}$$

und

$$K^- = \{C \setminus \{x_n\} \mid C \in K, \neg x_n \notin C\}.$$

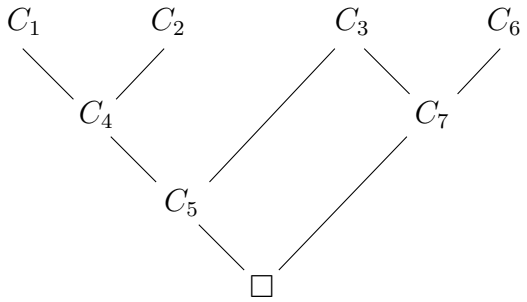
In keiner dieser beiden Klauselmengen kommt die Variable x_n noch vor. Es handelt sich also bei K^+ und K^- um Klauselmengen, in denen nur die Variablen $x_1, \ldots, x_{n-1}$ vorkommen.

Wir beobachten zunächst, dass keine der beiden Klauselmengen K^+ und K^- erfüllbar sein kann. Wir zeigen dies mit einem Widerspruchsbeweis. Wäre K^+ erfüllbar, so gäbe es eine Bewertung B der Variablen $x_1, \ldots, x_{n-1}$ mit $[\![K^+]\!]_B = 1$. Diese Bewertung können wir zu einer Bewertung B' der Variablen $x_1, \ldots, x_n$ erweitern, indem wir die Werte von $x_1, \ldots, x_{n-1}$ übernehmen und die Variable x_n auf den Wert 1 setzen. Wegen $[\![K^+]\!]_B = 1$ gilt auch für diese Erweiterung $[\![K^+]\!]_{B'} = 1$. Das bedeutet, die Bewertung B' erfüllt alle Klauseln aus K, die das Literal x_n nicht enthalten. Sie erfüllt aber auch alle Klauseln aus K, die das Literal x_n enthalten, da $B'(x_n) = 1$ gilt. Somit ist B' eine erfüllende Belegung für K im Widerspruch dazu, dass K unerfüllbar ist. Analog kann man argumentieren, dass auch die Klauselmenge K^- unerfüllbar ist.

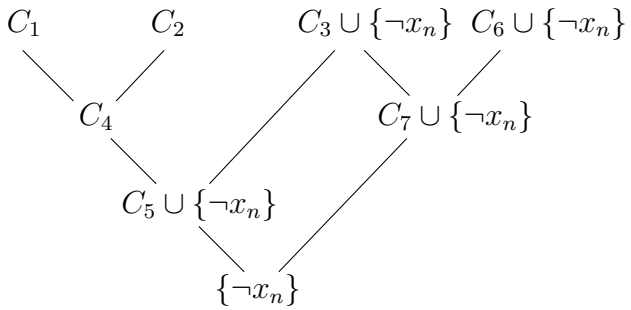
Wir haben nun mit K^+ und K^- zwei unerfüllbare Klauselmengen, in denen nur die Variablen $x_1, \ldots, x_{n-1}$ vorkommen. Die Induktionsvoraussetzung besagt, dass $\square \in$

$\text{Res}^*(K^+)$ und $\square \in \text{Res}^*(K^-)$ gilt. Es sei $C_1, C_2, \ldots, C_m$ eine Ableitung der Klausel $\square$ in K^+ im Resolutionskalkül. Es gelte also $C_m = \square$ und für jedes $i \in \{1, \ldots, m\}$ sei C_i entweder eine Klausel aus K^+ oder die Resolvente zweier Klauseln C_j und C_k mit $j < i$ und $k < i$. Gemäß der Definition von K^+ gilt $C_i \in K$ oder $C_i \cup \{\neg x_n\} \in K$ für jede Klausel $C_i \in K^+$. Wir übertragen die Ableitung $C_1, \ldots, C_m$ von der Klauselmenge K^+ auf die Klauselmenge K, indem wir jede Klausel $C_i \in K^+$ durch die entsprechende Klausel aus K ersetzen. Wir passen auch die Resolventen dementsprechend an und fügen gegebenenfalls das Literal $\neg x_n$ ein.

Die folgende Abbildung zeigt ein Beispiel für eine Ableitung $C_1, \ldots, C_m$ der Klausel $\square$ in K^+.



Sei $C_1, C_2 \in K$ und $C_3 \cup \{\neg x_n\}, C_6 \cup \{\neg x_n\} \in K$, so entspricht $C_1, \ldots, C_m$ der folgenden Ableitung in K.



Je nachdem, ob das Literal $\neg x_n$ ergänzt wurde oder nicht, erhalten wir aus $C_1, \ldots, C_m$ entweder eine Ableitung der Klausel $\square$ in K oder eine Ableitung der Klauselmenge $\{\neg x_n\}$ in K. Man kann analog argumentieren, dass man mithilfe der Klauselmenge K^- entweder eine Ableitung der Klausel $\square$ in K oder eine Ableitung der Klauselmenge $\{x_n\}$ in K konstruieren kann.

Gibt es eine Ableitung von $\square$ in K, so ist der Beweis abgeschlossen, denn dies ist gleichbedeutend mit $\square \in \text{Res}^*(K)$. Ansonsten gibt es Ableitungen der Klauselmengen $\{x_n\}$ und $\{\neg x_n\}$ in K. Sind diese Klauselmengen erzeugt, so kann auch auf sie die Resolution angewendet werden, was die leere Klausel $\square$ ergibt. Also gilt auch in diesem Fall $\square \in \text{Res}^*(K)$. $\square$

Es ergibt sich der folgende Algorithmus zum Test, ob eine gegebene Formel $\varphi \in \text{AL}$ erfüllbar ist.

Resolution(φ)

1. $R := \mathcal{K}(\varphi)$;
2. **while** $(R \neq \mathrm{Res}(R))$ { $R := \mathrm{Res}(R)$; }
3. **if** $(\square \in R)$ { **return** „φ unerfüllbar“; } **else** { **return** „φ erfüllbar“; }

Zwei ganz wesentliche Eigenschaften eines Kalküls sind *Korrektheit* und *Vollständigkeit*. Ein Kalkül heißt *korrekt*, wenn darin nur wahre Aussagen ableitbar sind, und er heißt *vollständig*, wenn darin alle wahren Aussagen ableitbar sind. Im Idealfall möchte man einen Kalkül entwerfen, der sowohl korrekt als auch vollständig ist. Theorem 5.16 besagt, dass dies für den Resolutionskalkül der Fall ist. Der obige Algorithmus gibt für jede erfüllbare Formel aus, dass sie erfüllbar ist, und er gibt für keine unerfüllbare Formel aus, dass sie erfüllbar ist.

Für viele Formeln ist der Algorithmus recht effizient und deutlich schneller als der triviale Algorithmus, der für alle Bewertungen der Variablen testet, ob sie die Formel erfüllen. Im Allgemeinen benötigt aber auch der obige Algorithmus exponentielle Laufzeit. Da das Erfüllbarkeitsproblem für aussagenlogische Formeln NP-schwer ist (vergleiche Kapitel 1), gibt es vermutlich auch gar keinen effizienten Algorithmus für dieses Problem. Dies werden wir im vierten Semester ausführlich diskutieren.

5.2 Prädikatenlogik

Viele Sachverhalte, denen wir in der Mathematik und der theoretischen Informatik begegnen, können in der Aussagenlogik nicht dargestellt werden. Schon einfache Aussagen wie „es gibt keine reelle Zahl, deren Quadrat negativ ist“ oder „die Summe von zwei ungeraden Zahlen ist gerade, 3 und 5 sind ungerade Zahlen, also ist $3 + 5$ gerade“ lassen sich nicht formulieren. Neben dem Fehlen von Quantoren liegt dies daran, dass wir in der Aussagenlogik nur auf Aussagenvariablen zurückgreifen können, die entweder wahr oder falsch sind. Wir haben jedoch keine Möglichkeit, Aussagen über die Elemente einer Struktur wie zum Beispiel über reelle Zahlen zu treffen. Wir lernen in diesem Abschnitt die Prädikatenlogik kennen, die die Aussagenlogik um diese Möglichkeit und um Quantoren erweitert.

5.2.1 Signaturen und Strukturen

In der Prädikatenlogik treffen wir Aussagen über die Elemente von Strukturen. Wir formalisieren zunächst, was wir unter einer Struktur verstehen, und führen bereits an dieser Stelle eine Trennung zwischen Syntax und Semantik ein. Eine Struktur ist eine Menge (zum Beispiel die Menge der reellen Zahlen) gemeinsam mit Funktionen (zum Beispiel $+$ und $\cdot$) und Relationen (zum Beispiel $\leq$), die auf dieser Menge definiert sind. Darüber hinaus können in einer Struktur bestimmte Elemente explizit ausgezeichnet sein (zum Beispiel die neutralen Elemente der Addition und Multiplikation 0 und 1). Diese Elemente nennen wir auch Konstanten.

Die Syntax einer Struktur wird durch ihre *Signatur* beschrieben. Ähnlich wie bei einem Interface in Java enthält die Signatur nur Informationen darüber, welche Art von Funktionen, Relationen und Konstanten es gibt und wie diese heißen, nicht jedoch wie sie konkret definiert sind. In dem obigen Beispiel sagt uns die Signatur, dass wir über eine Menge reden, auf der zwei zweistellige Funktionen $+$ und $\cdot$ sowie eine zweistellige Relation $\leq$ definiert sind und in der zwei Konstanten 0 und 1 ausgezeichnet sind.

Definition 5.17. *Eine* Signatur $\sigma = \{f_1, \ldots, f_k, R_1, \ldots, R_\ell, c_1, \ldots, c_m\}$ *ist eine Menge von Funktionssymbolen* $f_1, \ldots, f_k$*, Relationssymbolen* $R_1, \ldots, R_\ell$ *und Konstantensymbolen* $c_1, \ldots, c_m$*. Jedes Funktionssymbol* $f \in \sigma$ *und Relationssymbol* $R \in \sigma$ *hat eine* Stelligkeit $\mathrm{ar}(f) \in \mathbb{N}$ *bzw.* $\mathrm{ar}(R) \in \mathbb{N}$*.*

Beispiele

- Die Menge $\{\bar{+}, \bar{\cdot}, \bar{0}, \bar{1}\}$ ist eine Signatur, wobei $\bar{+}$ und $\bar{\cdot}$ Symbole für zweistellige Funktionen (d. h. $\mathrm{ar}(\bar{+}) = \mathrm{ar}(\bar{\cdot}) = 2$) und $\bar{0}$ und $\bar{1}$ Konstantensymbole sind.
- Die Menge $\{\bar{+}, \bar{\cdot}, \bar{\leq}, \bar{0}, \bar{1}\}$ mit $\mathrm{ar}(\bar{+}) = \mathrm{ar}(\bar{\cdot}) = \mathrm{ar}(\bar{\leq}) = 2$ ist eine Signatur mit zwei zweistelligen Funktionssymbolen $\bar{+}$ und $\bar{\cdot}$, einem zweistelligen Relationssymbol $\bar{\leq}$ und zwei Konstantensymbolen $\bar{0}$ und $\bar{1}$.

Eine Signatur beschreibt ausschließlich die Syntax einer Struktur. Um die Semantik zu beschreiben, muss die Menge, um die es geht, spezifiziert werden und die Funktionen, Relationen und Konstanten müssen definiert werden.

Definition 5.18. *Es sei* σ *eine Signatur. Eine* σ-Struktur $\mathfrak{A} = (A, \alpha)$ *ist ein Paar bestehend aus einer nichtleeren Menge* A*, dem sogenannten* Universum*, und einer* Interpretationsfunktion α*. Die Funktion* α *ordnet*

- *jedem Funktionssymbol* $f \in \sigma$ *eine* $\mathrm{ar}(f)$*-stellige Funktion* $f^{\mathfrak{A}} \colon A^{\mathrm{ar}(f)} \to A$*,*
- *jedem Relationssymbol* $R \in \sigma$ *eine* $\mathrm{ar}(R)$*-stellige Relation* $R^{\mathfrak{A}} \subseteq A^{\mathrm{ar}(R)}$
- *und jedem Konstantensymbol* $c \in \sigma$ *ein Element* $c^{\mathfrak{A}} \in A$ *zu.*

Beispiel

Wir betrachten die Signatur $\{\bar{+}, \bar{\cdot}, \bar{0}, \bar{1}\}$ aus dem obigen Beispiel. Diese Signatur besagt, dass wir Aussagen über eine Menge treffen wollen, auf der zwei zweistellige Funktionen definiert und in der zwei Konstanten ausgezeichnet sind.

Das Paar $(\mathbb{R}, \alpha)$ ist eine σ-Struktur, wenn die Abbildung α die Funktionssymbole $\bar{+}$ und $\bar{\cdot}$ auf die normale Addition und Multiplikation in $\mathbb{R}$ abbildet und wenn $\alpha(\bar{0}) = 0$ und $\alpha(\bar{1}) = 1$ gilt.

Ebenso ist für $n \in \mathbb{N}$ das Paar $(\mathbb{Z}/n\mathbb{Z}, \alpha)$ eine σ-Struktur, wenn die Abbildung α die Funktionssymbole $\bar{+}$ und $\bar{\cdot}$ auf die Verknüpfungen $\oplus_n$ und $\odot_n$ abbildet und wenn $\alpha(\bar{0}) = [\![0]\!]_n$ und $\alpha(\bar{1}) = [\![1]\!]_n$ gilt.

Dieses Beispiel lässt bereits den Nutzen von Signaturen erahnen. Die Syntax prädikatenlogischer Formeln werden wir nur basierend auf einer gegebenen Signatur σ

definieren. Lediglich die Semantik hängt von der konkreten σ-Struktur ab. Auf diese Weise erfolgt eine strikte Trennung von Syntax und Semantik.

Um den Sinn von Signaturen und Strukturen noch weiter zu erläutern, betrachten wir *relationale Datenbanken*. Dies ist ein wichtiges Thema der Informatik, für das mathematische Logik von grundlegender Bedeutung ist. Grob gesprochen ist eine relationale Datenbank eine endliche Menge von endlichen Tabellen. Jede Zeile einer Tabelle R ist ein Tupel $(a_1, \ldots, a_n) \in D_1 \times \cdots \times D_n$, wobei die Menge D_i den Wertebereich der Einträge in Spalte i angibt. Die Tabelle R kann demnach als n-stellige Relation über der Menge $D = D_1 \cup \cdots \cup D_n$ aufgefasst werden, also $R \subseteq D^n$.

Besteht die Datenbank aus m Tabellen, so kann ihr aktueller Inhalt durch Relationen $R_1, \ldots, R_m$ über einer entsprechenden Menge D beschrieben werden. Der aktuelle Inhalt der Datenbank bildet also eine Struktur $(D, R_1, \ldots, R_m)$. Die zugehörige Signatur enthält nur Informationen darüber, wie viele Relationen (d. h. wie viele Tabellen) es gibt und wie deren Stelligkeiten (d. h. die Anzahl an Spalten) gewählt sind. Während die Struktur den Inhalt der Tabelle beschreibt, enthält die Signatur also nur Informationen über die Syntax.

5.2.2 Syntax

Um die Syntax der Prädikatenlogik zu beschreiben, fixieren wir eine beliebige Signatur σ. Die Formeln, die basierend auf dieser Signatur gebildet werden können, nennen wir σ-Formeln. Sie bestehen aus Zeichen des Alphabets $\mathrm{Alph}(\sigma)$, das die folgenden Elemente enthält:

- die Funktionssymbole, Relationssymbole und Konstantensymbole aus σ,
- eine abzählbar unendliche Menge von Variablen $\mathrm{VAR} = \{x_1, x_2, x_3, \ldots\}$,
- das Gleichheitszeichen $=$,
- die Verknüpfungen $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$,
- die Quantoren $\exists$ und $\forall$,
- die Klammern (und) sowie das Komma.

Bevor wir angeben, wie σ-Formeln gebildet werden, definieren wir die Menge der sogenannten σ-Terme. Diese bilden einen wesentlichen Bestandteil von σ-Formeln.

Definition 5.19. *Die Menge $T(\sigma)$ der* σ-Terme *ist die kleinste Sprache über dem Alphabet* $\mathrm{Alph}(\sigma)$ *mit den folgenden Eigenschaften.*

a) Es gilt $\mathrm{VAR} \subseteq T(\sigma)$ *und jedes Konstantensymbol $c \in \sigma$ gehört zu $T(\sigma)$.*

b) Sind $t_1, \ldots, t_n \in T(\sigma)$ und ist $f \in \sigma$ ein n-stelliges Funktionssymbol, so ist auch $f(t_1, \ldots, t_n) \in T(\sigma)$.

Beispiele

Es sei $\sigma = \{f_1, f_2, R_1, c_1, c_2\}$ eine Signatur mit den Funktionssymbolen f_1 und f_2 mit $\mathrm{ar}(f_1) = 1$ und $\mathrm{ar}(f_2) = 3$, sowie dem Relationssymbol R_1 mit $\mathrm{ar}(R_1) = 2$ und den Konstantensymbolen c_1 und c_2.

Die folgenden Wörter sind σ-Terme: c_1, $f_1(x_2)$, $f_2(c_2, x_1, c_2)$, $f_1(f_1(x_1))$, $f_2(f_1(c_1), x_2, c_1)$, $f_1(f_2(x_1, f_1(c_1), x_2))$.

Mithilfe von σ-Termen können wir nun die Syntax der Prädikatenlogik definieren.

Definition 5.20. *Die Menge* $\mathrm{FO}(\sigma)$ *der* σ-Formeln *der Prädikatenlogik (FO steht für die englische Bezeichnung first-order logic) ist die kleinste Sprache über dem Alphabet* $\mathrm{Alph}(\sigma)$ *mit den folgenden Eigenschaften.*

a) Sind $t_1, t_2 \in T(\sigma)$*, so gehört das Wort* $t_1 = t_2$ *zu* $\mathrm{FO}(\sigma)$*.*

b) Sind $t_1, \ldots, t_n \in T(\sigma)$ *und ist* $R \in \sigma$ *ein n-stelliges Relationssymbol, so gehört das Wort* $R(t_1, \ldots, t_n)$ *zu* $FO(\sigma)$*.*

c) Sind $\varphi_1 \in \mathrm{FO}(\sigma)$ *und* $\varphi_2 \in \mathrm{FO}(\sigma)$*, so gehören auch die Wörter* $\neg\varphi_1$*,* $(\varphi_1 \wedge \varphi_2)$*,* $(\varphi_1 \vee \varphi_2)$*,* $(\varphi_1 \to \varphi_2)$ *und* $(\varphi_1 \leftrightarrow \varphi_2)$ *zu* $\mathrm{FO}(\sigma)$*.*

d) Ist $\varphi \in \mathrm{FO}(\sigma)$ *und* $x \in \mathrm{VAR}$*, so gehören auch die Wörter* $\exists x\varphi$ *und* $\forall x\varphi$ *zu* $\mathrm{FO}(\sigma)$*.*

Beispiele

Es sei $\sigma = \{f_1, f_2, R_1, c_1, c_2\}$ eine Signatur mit den Funktionssymbolen f_1 und f_2 mit $\mathrm{ar}(f_1) = 1$ und $\mathrm{ar}(f_2) = 3$, sowie dem Relationssymbol R_1 mit $\mathrm{ar}(R_1) = 2$ und den Konstantensymbolen c_1 und c_2.

Die folgenden Wörter sind Formeln aus $\mathrm{FO}(\sigma)$:

- $R_1(c_1, x_2)$,
- $f_1(x_2) = c_1$,
- $\exists x_1 f_1(x_2) = c_1$,
- $\forall x_2 f_1(x_2) = c_1$,
- $(x_1 = x_2 \leftrightarrow (f_1(x_1) = f_2(x_2, c_1, c_2) \vee x_1 = c_1))$,
- $\forall x_1 \forall x_2 (x_1 = x_2 \leftrightarrow f_1(x_1) = f_1(x_2))$,
- $(\exists x_1 f_1(x_1) = c_1 \to \forall x_1 R_1(x_1, c_2))$.

Variablen können in σ-Formeln entweder *frei* oder *gebunden* vorkommen. Das Vorkommen einer Variablen x ist gebunden, falls es in einer Unterformel der Form $\exists x\varphi$ oder $\forall x\varphi$ auftritt, anderenfalls ist es frei. Um dies zu illustrieren, betrachten wir wieder die Signatur σ aus dem obigen Beispiel und die folgende σ-Formel, in der wir genau die gebunden Vorkommen von Variablen unterstrichen haben:

$$\forall \underline{x_1}(f_1(\underline{x_1}) = x_2 \vee \exists \underline{x_2} R_1(\underline{x_1}, \underline{x_2})).$$

Dieses Beispiel zeigt, dass eine Variable in derselben Formel an verschiedenen Stellen gebunden und frei vorkommen kann. Für eine Formel $\varphi \in \mathrm{FO}(\sigma)$ und einen Term $t \in T(\sigma)$ bezeichnen wir im Folgenden mit $\mathrm{Var}(\varphi)$ bzw. $\mathrm{Var}(t)$ die Menge der Variablen, die in φ bzw. t mindestens einmal vorkommen. Außerdem bezeichnen wir für eine Formel φ mit $\mathrm{frei}(\varphi)$ die Menge der Variablen, die in φ mindestens einmal frei vorkommen. Es gilt also beispielsweise

$$\mathrm{frei}(\exists x_1 R_1(x_1, c_1)) = \emptyset \quad \text{und} \quad \mathrm{frei}(\forall x_1(f_1(x_1) = x_2 \vee \exists x_2 R_1(x_1, x_2))) = \{x_2\}.$$

Ist $\varphi \in \mathrm{FO}(\sigma)$ mit $\mathrm{frei}(\varphi) = \{x_1, \ldots, x_k\}$, so schreiben wir oft $\varphi(x_1, \ldots, x_k)$, um die Menge der freien Variablen zu kennzeichnen. Eine Formel $\varphi \in \mathrm{FO}(\sigma)$ ohne freie Variablen nennen wir auch einen *σ-Satz*.

5.2.3 Semantik

Bislang haben wir nur die Syntax der Prädikatenlogik definiert. Da wir bereits in den vergangenen Kapiteln oft Ausdrücke benutzt haben, die große Ähnlichkeit mit prädikatenlogischen Formeln aufweisen, hat der Leser aber sicherlich schon eine gewisse Erwartung, was prädikatenlogische Formeln bedeuten. Wir werden die Semantik nun formal definieren.

Definition 5.21. *Es sei σ eine Signatur. Eine* σ-Interpretation $J = (\mathfrak{A}, \beta)$ *ist ein Paar bestehend aus einer σ-Struktur $\mathfrak{A} = (A, \alpha)$ und einer Abbildung $\beta\colon X \to A$ für eine Menge $X \subseteq \mathrm{VAR}$.*

Eine solche σ-Interpretation ordnet jedem Term $t \in T(\sigma)$ mit $\mathrm{Var}(t) \subseteq X$ einen Wert $[\![t]\!]_J \in A$ und jeder Formel $\varphi \in \mathrm{FO}(\sigma)$ mit $\mathrm{Var}(\varphi) \subseteq X$ einen Wahrheitswert $[\![\varphi]\!]_J \in \{0, 1\}$ zu. Wir sagen, dass J zu einem Term $t \in T(\sigma)$ oder einer Formel $\varphi \in \mathrm{FO}(\sigma)$ passt, *wenn $\mathrm{Var}(t) \subseteq X$ bzw. $\mathrm{Var}(\varphi) \subseteq X$ gilt.*

Der Wert eines Terms $t \in T(\sigma)$ mit $\mathrm{Var}(t) \subseteq X$ ist induktiv durch die folgenden Regeln definiert.

- *Für eine Variable $x \in X$ gilt $[\![x]\!]_J = \beta(x)$.*
- *Für ein Konstantensymbol $c \in \sigma$ gilt $[\![c]\!]_J = c^{\mathfrak{A}}$.*
- *Sind $t_1, \ldots, t_n \in T(\sigma)$ und ist $f \in \sigma$ ein n-stelliges Funktionssymbol, so gilt $[\![f(t_1, \ldots, t_n)]\!]_J = f^{\mathfrak{A}}([\![t_1]\!]_J, \ldots, [\![t_n]\!]_J)$.*

Der Wert einer Formel $\varphi \in \mathrm{FO}(\sigma)$ mit $\mathrm{Var}(\varphi) \subseteq X$ ist induktiv durch die folgenden Regeln definiert.

- *Für zwei Terme $t_1, t_2 \in T(\sigma)$ gilt*

$$[\![t_1 = t_2]\!]_J = \begin{cases} 1 & \text{falls } [\![t_1]\!]_J = [\![t_2]\!]_J, \\ 0 & \text{sonst.} \end{cases}$$

- *Für $t_1, \dots, t_n \in T(\sigma)$ und ein n-stelliges Relationssymbol $R \in \sigma$ gilt*

$$[\![R(t_1, \dots, t_n)]\!]_J = \begin{cases} 1 & \text{falls } ([\![t_1]\!]_J, \dots, [\![t_n]\!]_J) \in R^{\mathfrak{A}}, \\ 0 & \text{sonst.} \end{cases}$$

- *Sind $\varphi_1 \in \mathrm{FO}(\sigma)$ und $\varphi_2 \in \mathrm{FO}(\sigma)$ und hat φ die Gestalt $\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \to \varphi_2)$ oder $(\varphi_1 \leftrightarrow \varphi_2)$, so ist der Wahrheitswert von φ genauso definiert wie in der Aussagenlogik, d. h.*

$$\begin{aligned} [\![\neg\varphi_1]\!]_J &= 1 - [\![\varphi_1]\!]_J, \\ [\![\varphi_1 \wedge \varphi_2]\!]_J &= \min\{[\![\varphi_1]\!]_J, [\![\varphi_2]\!]_J\}, \\ [\![\varphi_1 \vee \varphi_2]\!]_J &= \max\{[\![\varphi_1]\!]_J, [\![\varphi_2]\!]_J\}, \\ [\![\varphi_1 \to \varphi_2]\!]_J &= [\![(\neg\varphi_1 \vee \varphi_2)]\!]_J, \\ [\![\varphi_1 \leftrightarrow \varphi_2]\!]_J &= \begin{cases} 1 & \text{falls } [\![\varphi_1]\!]_J = [\![\varphi_2]\!]_J, \\ 0 & \text{sonst.} \end{cases} \end{aligned}$$

- *Für eine Abbildung $\beta\colon X \to A$, eine Variable $x \in \mathrm{VAR}$ und ein $a \in A$ sei $\beta[x/a]\colon X \cup \{x\} \to A$ mit*

$$\beta[x/a](y) = \begin{cases} \beta(y) & \text{falls } y \neq x, \\ a & \text{falls } y = x. \end{cases}$$

  Die Abbildung $\beta[x/a]$ setzt die Variable x also auf den Wert a, egal ob der Wert von x bereits in β gesetzt wurde oder nicht. Abgesehen davon stimmt die Belegung $\beta[x/a]$ mit der Belegung β überein. Mit $J[x/a]$ bezeichnen wir die σ-Interpretation $(\mathfrak{A}, \beta[x/a])$.

  Ist $\varphi \in \mathrm{FO}(\sigma)$ und $x \in \mathrm{VAR}$, so gilt

$$[\![\exists x\varphi]\!]_J = \max_{a \in A} [\![\varphi]\!]_{J[x/a]}$$

  und

$$[\![\forall x\varphi]\!]_J = \min_{a \in A} [\![\varphi]\!]_{J[x/a]}.$$

Gilt $[\![\varphi]\!]_J = 1$, so sagen wir, dass die Formel φ für die Interpretation J wahr *ist. Gilt hingegen $[\![\varphi]\!]_J = 0$, so sagen wir, dass die Formel φ für die Interpretation J* falsch *ist.*

Genau wie in der Aussagenlogik ist es auch in der Prädikatenlogik oft wichtig, zu entscheiden, ob eine gegebene Formel erfüllbar ist, oder ob sie sogar für jede Interpretation wahr ist.

Definition 5.22. *Es sei σ eine Signatur und es sei $\varphi \in \mathrm{FO}(\sigma)$ eine prädikatenlogische Formel. Ferner sei $J = (\mathfrak{A}, \beta)$ eine* σ-Interpretation, *die zu φ passt.*

a) *Die Interpretation J heißt* Modell von φ, *wenn $[\![\varphi]\!]_J = 1$ gilt. Wir sagen dann, dass die Interpretation J die Formel φ* erfüllt *und schreiben $J \models \varphi$. Wir nennen J ein* Modell der Formelmenge $\Phi \subseteq \mathrm{FO}(\sigma)$, *wenn J alle Formeln aus Φ erfüllt (d. h. insbesondere, dass J zu allen Formeln aus Φ passt) und schreiben dann $J \models \Phi$.*

b) *Die Formel φ heißt* erfüllbar, *wenn es ein Modell für sie gibt, d. h. eine zu ihr passende Interpretation J mit $[\![\varphi]\!]_J = 1$.*

c) *Die Formel φ heißt* gültig, *wenn $[\![\varphi]\!]_J = 1$ für jede zu ihr passende Interpretation J gilt.*

Ist $J = (\mathfrak{A}, \beta)$ eine Interpretation für einen σ-Satz φ, also eine σ-Formel ohne freie Variablen, so ist die Bewertung β nicht relevant für den Wahrheitswert $[\![\varphi]\!]_J$. Deshalb schreiben wir statt $[\![\varphi]\!]_J$ auch $[\![\varphi]\!]_{\mathfrak{A}}$. Diese Beobachtung beweisen wir aus Zeitgründen nicht in der Vorlesung. Ist $J = (\mathfrak{A}, \beta)$ ein Modell für einen σ-Satz φ, so sagen wir auch, dass $\mathfrak{A}$ ein Modell von φ ist und schreiben dementsprechend $\mathfrak{A} \models \varphi$.

Ebenso wie in der Aussagenlogik werden wir von der strikten Syntax der Prädikatenlogik abweichen, um die Lesbarkeit von Termen und Formeln zu verbessern. Wir werden auf analoge Weise Klammern weglassen und von den vorgegebenen Variablennamen abweichen. Gelegentlich schreiben wir auch $\exists x : \varphi$ statt $\exists x \varphi$ und $\forall x : \varphi$ statt $\forall x \varphi$, wenn dies der Übersichtlichkeit dient. Möchten wir ausdrücken, dass zwei Formeln φ_1 und φ_2 Zeichen für Zeichen übereinstimmen, so schreiben wir $\varphi_1 \doteq \varphi_2$, da das normale Gleichheitszeichen bereits in der Syntax der Prädikatenlogik enthalten ist.

Beispiele

- Wir betrachten die Signatur $\sigma = \{f\}$, die aus einem einstelligen Funktionssymbol f besteht, und die folgenden Sätze aus $\mathrm{FO}(\sigma)$:

$$\begin{aligned} \varphi_1 &\doteq (\forall y \exists x : f(x) = y), \\ \varphi_2 &\doteq (\forall x \exists y : f(x) = y), \\ \varphi_3 &\doteq (\exists x \exists y \exists z : f(x) = y \wedge f(x) = z \wedge \neg(y = z)). \end{aligned}$$

  Der Satz φ_1 ist erfüllbar, aber nicht gültig. Er besagt, dass die Funktion f surjektiv ist. Die σ-Struktur $\mathfrak{A} = (\mathbb{N}, \alpha)$ ist für die Funktion $f^{\mathfrak{A}} \colon \mathbb{N} \to \mathbb{N}$ mit $f^{\mathfrak{A}}(n) = n$ ein Modell von φ_1. Für die Funktion $f^{\mathfrak{A}} \colon \mathbb{N} \to \mathbb{N}$ mit $f^{\mathfrak{A}}(n) = 1$ ist $\mathfrak{A}$ hingegen kein Modell von φ_1.

  Der Satz φ_2 ist gültig. Jede σ-Struktur $\mathfrak{A} = (A, \alpha)$ mit einer beliebigen Funktion $f^{\mathfrak{A}} \colon A \to A$ erfüllt φ_2, da eine Funktion per Definition jedes Element des Definitionsbereichs auf ein Element der Zielmenge abbildet.

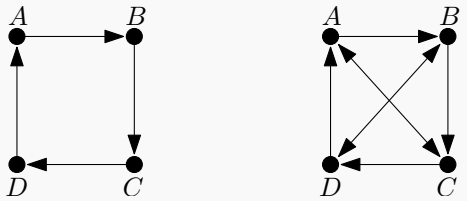
  Der Satz φ_3 ist nicht erfüllbar. Dies ist eine einfache Folgerung daraus, dass eine Funktion jedes Element des Definitionsbereichs auf genau ein Element der Zielmenge abbildet.

- Sei nun $\sigma = \{R\}$ eine Signatur, die aus einem zweistelligen Relationssymbol R besteht. Wir betrachten die folgenden Formeln aus FO(σ):

$$\begin{aligned} \varphi_1 &\doteq (\forall x \forall y : R(x,y) \rightarrow R(y,x)), \\ \varphi_2(x,y) &\doteq (\exists z : R(x,y) \vee (R(x,z) \wedge R(z,y))), \\ \varphi_3 &\doteq (\forall x \forall y : \varphi_2(x,y)). \end{aligned}$$

  Der Satz φ_1 ist erfüllbar, aber nicht gültig. Er besagt, dass die Relation R symmetrisch ist.

  Für die Formel φ_2 mit den freien Variablen x und y betrachten wir die σ-Struktur $\mathfrak{A} = (\{A, B, C, D\}, \alpha)$, wobei $R^{\mathfrak{A}}$ die Relation ist, die im Bild unten links dargestellt ist.



  Da es sich bei φ_2 um eine Formel mit freien Variablen handelt, gehört zu einer Interpretation J neben der Struktur $\mathfrak{A}$ noch eine Bewertung β. Für die Bewertung β mit $\beta(x) = A$ und $\beta(y) = C$ gilt $[\![\varphi_2]\!]_J = 1$, denn es gilt $(A, B) \in R^{\mathfrak{A}}$ und $(B, C) \in R^{\mathfrak{A}}$. Für die Bewertung β mit $\beta(x) = A$ und $\beta(y) = D$ gilt hingegen $[\![\varphi_2]\!]_J = 0$. Allgemein können wir festhalten, dass die Interpretation J die Formel φ_2 genau dann erfüllt, wenn $\beta(x)$ zu $\beta(y)$ in Relation $R^{\mathfrak{A}}$ steht oder wenn es in dem Diagramm der Relation $R^{\mathfrak{A}}$ einen Weg von $\beta(x)$ zu $\beta(y)$ über ein Element z gibt.

  Da es sich bei φ_3 um einen Satz handelt, ist die Belegung β irrelevant. Die Struktur $\mathfrak{A} = (\{A, B, C, D\}, \alpha)$ mit der Relation $R^{\mathfrak{A}}$, die oben links dargestellt ist, ist kein Modell für φ_3. Ist $R^{\mathfrak{A}}$ hingegen die Relation, die oben rechts dargestellt ist, so ist $\mathfrak{A} = (\{A, B, C, D\}, \alpha)$ ein Modell für φ_3.

Wir haben oben bereits die Verbindung zwischen Prädikatenlogik und relationalen Datenbanken angedeutet. Nun betrachten wir noch kurz die Datenbanksprache SQL, die den meisten Lesern vermutlich vertraut ist. Wir gehen davon aus, dass eine Datenbank mit einer Tabelle *Student* gegeben ist, in der Informationen über die Hörer dieser Vorlesung eingetragen sind. Die Spalten seien *Name*, *Note* und *Tutor* mit der offensichtlichen Bedeutung. Die Anfrage

„Gib die Namen aller Studenten aus, die die Note 1,0 haben."

lässt sich in SQL wie folgt formulieren.

```
SELECT Name
```

FROM Student
WHERE Note = ‘1.0’

Wie oben beschrieben können wir die Tabelle als eine 3-stellige Relation R auffassen. Damit entspricht die obige SQL-Abfrage der prädikatenlogischen Formel

$$\varphi(x_{\text{Name}}) \doteq (\exists x_{\text{Tutor}} : R(x_{\text{Name}}, 1.0, x_{\text{Tutor}})).$$

Die SQL-Abfrage liefert alle Belegungen für die freie Variable x_{Name}, die die Formel $\varphi(x_{\text{Name}})$ erfüllen. Analog entspricht die Anfrage

„Gib die Namen aller Studenten aus, die die Note 1,0 und nicht den Tutor ‘Meier’ haben.“

der SQL-Abfrage

SELECT Name
FROM Student
WHERE Note = ‘1.0’ AND NOT Tutor = ‘Meier’

Diese entspricht wiederum der prädikatenlogischen Formel

$$\varphi(x_{\text{Name}}) \doteq (\exists x_{\text{Tutor}} : R(x_{\text{Name}}, 1.0, x_{\text{Tutor}}) \wedge \neg(x_{\text{Tutor}} = \text{‘Meier’})).$$

5.2.4 Ausblick

Viele interessante Aspekte und Fragestellungen im Rahmen der Prädikatenlogik können wir aus Zeitgründen in dieser Vorlesung nicht mehr im Detail besprechen. Auf einige davon gehen wir in diesem Abschnitt noch kurz ein.

Es ist eine natürliche Frage, wo die Grenzen der Prädikatenlogik liegen. Man stellt schnell fest, dass es gewisse alltägliche mathematische Sachverhalte gibt, die auch mithilfe der Prädikatenlogik nicht formuliert werden können. Dazu betrachten wir noch einmal ein Beispiel aus dem vorherigen Abschnitt. Dort haben wir für die Signatur $\sigma = \{R\}$ mit einem binären Relationssymbol R einen Satz φ_3 konstruiert, der von einer Interpretation $J = (\mathfrak{A}, \beta)$ mit $\mathfrak{A} = (A, \alpha)$ genau dann erfüllt wird, wenn für alle $x, y \in A$ gilt: x und y stehen in Relation $R^{\mathfrak{A}}$ oder in dem Diagramm der Relation $R^{\mathfrak{A}}$ gibt es einen Weg von x nach y über ein beliebiges Element z.

Gibt es auch einen σ-Satz φ, der von einer Interpretation $J = (\mathfrak{A}, \beta)$ mit $\mathfrak{A} = (A, \alpha)$ genau dann erfüllt wird, wenn für alle $x, y \in A$ gilt: es gibt in dem Diagramm der Relation $R^{\mathfrak{A}}$ einen Weg von x nach y beliebiger Länge? Nach einigen vergeblichen Versuchen, einen solchen Satz aufzustellen, gewinnt man die Intuition, dass dies in der Prädikatenlogik, die wir kennengelernt haben, nicht möglich ist, was auch tatsächlich formal gezeigt werden kann.

Grob gesprochen liegt das Problem darin begründet, dass es nicht möglich ist, Quantoren auf Relationen anzuwenden. Darf man Aussagen wie „für alle Relationen R gilt ...“ oder „es gibt eine Relation R, für die gilt ...“ bilden, so kann ein Satz mit der oben genannten Eigenschaft gebildet werden. Erlaubt man diese Quantifizierung über Relationen, so erhält man die *Prädikatenlogik zweiter Stufe*, die ausdrucksstärker als die Prädikatenlogik ist, die wir kennengelernt haben. Diese nennt man auch *Prädikatenlogik erster Stufe*.

Eine weitere Frage, die sich an unsere Diskussion der Aussagenlogik anschließt, ist, ob man algorithmisch entscheiden kann, ob eine gegebene Formel der Prädikatenlogik erfüllbar ist. Bei der Aussagenlogik war dies einfach, da nur endlich viele Bewertungen der Variablen getestet werden müssen. Die Prädikatenlogik hingegen trifft Aussagen über Strukturen und selbst für einfache Signaturen σ gibt es überabzählbar viele σ-Strukturen. Ein einfaches Testen aller möglichen Interpretationen ist also unmöglich. Dies besagt aber natürlich nicht, dass es nicht ein anderes algorithmisches Verfahren gibt, um die Erfüllbarkeit von Formeln der Prädikatenlogik zu testen.

Tatsächlich gibt es einen Kalkül für die Prädikatenlogik, der aus einer Reihe von einfachen Axiomen und Umformungsregeln besteht. Mit diesem Kalkül kann ein Algorithmus entworfen werden, der alle gültigen Formeln der Prädikatenlogik nach und nach in einer beliebigen Reihenfolge ausgibt. Diesen Algorithmus nennen wir auch einen Aufzähler. Die Existenz eines solchen Aufzählers legt für eine gegebene Formel φ folgenden Erfüllbarkeitstest nahe: Starte den Aufzähler und lasse ihn die erfüllbaren Formeln der Prädikatenlogik nach und nach ausgeben. Sobald er φ ausgibt, stoppe den Algorithmus und gib aus, dass φ erfüllbar ist.

Auf diese Weise wird für jede erfüllbare Formel in endlich vielen Schritten ausgegeben, dass sie erfüllbar ist. Problematisch sind aber Formeln, die nicht erfüllbar sind. Da wir nicht wissen, in welcher Reihenfolge der Aufzähler die erfüllbaren Formeln ausgibt, können wir die Aufzählung nie abbrechen und der oben beschriebene Algorithmus terminiert nicht. Man kann nachweisen, dass es sich bei dem Erfüllbarkeitstest von prädikatenlogischen Formeln, um ein *unentscheidbares* Problem handelt, also um ein Problem, für das es keinen korrekten Algorithmus gibt, der auf jeder Eingabe nach endlich vielen Schritten terminiert. Mit dieser Thematik werden wir uns im vierten Semester noch ausführlich beschäftigen.

Literaturverzeichnis

[1] Norbert Blum: **Einführung in formale Sprachen, Berechenbarkeit, Informations- und Lerntheorie**. Oldenbourg, 2007.

[2] Norbert Blum: **Skript zur Vorlesung „Logik und diskrete Strukturen“**, Universität Bonn, Wintersemester 2013/14. http://theory.cs.uni-bonn.de/blum/Lehre/Archiv/WS1314/luds.html.

[3] Erich Grädel: **Skript zur Vorlesung „Mathematische Logik“**, RWTH Aachen, Sommersemester 2011. http://logic.rwth-aachen.de/files/MaLo-SS12/script.pdf.

[4] John E. Hopcroft, Rajeev Motwani und Jeffrey D. Ullman: **Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit**. Pearson Studium, 2011. ISBN 978-3868940824.

[5] Rolf Klein: **Skript zur Vorlesung „Logik und diskrete Strukturen“**, Universität Bonn, Wintersemester 2011/12. http://tizian.cs.uni-bonn.de/Lehre/Vorlesungen/LuDS-WS1112/index.html.

[6] Thoralf Räsch: **Skript zur Vorlesung „Logik und diskrete Strukturen“**, Universität Bonn, Wintersemester 2009/10. http://www.math.uni-bonn.de/people/raesch/Papers_and_Notes/ThR_LuDS_Skript.pdf.

[7] Nicole Schweikardt: **Skript zur Vorlesung „Diskrete Modellierung“**, Goethe-Universität Frankfurt am Main, Wintersemester 2012/13. http://www.tks.informatik.uni-frankfurt.de/teaching/dismod/skript.

[8] Uwe Schöning: **Logik für Informatiker**. Spektrum Akademischer Verlag, 2000. ISBN 978-3827410054.

[9] Uwe Schöning: **Theoretische Informatik - kurz gefasst**. Spektrum Akademischer Verlag, 2008. ISBN 978-3827418241.

[10] Ingo Wegener: **Theoretische Informatik - eine algorithmenorientierte Einführung**. Teubner, 2005.