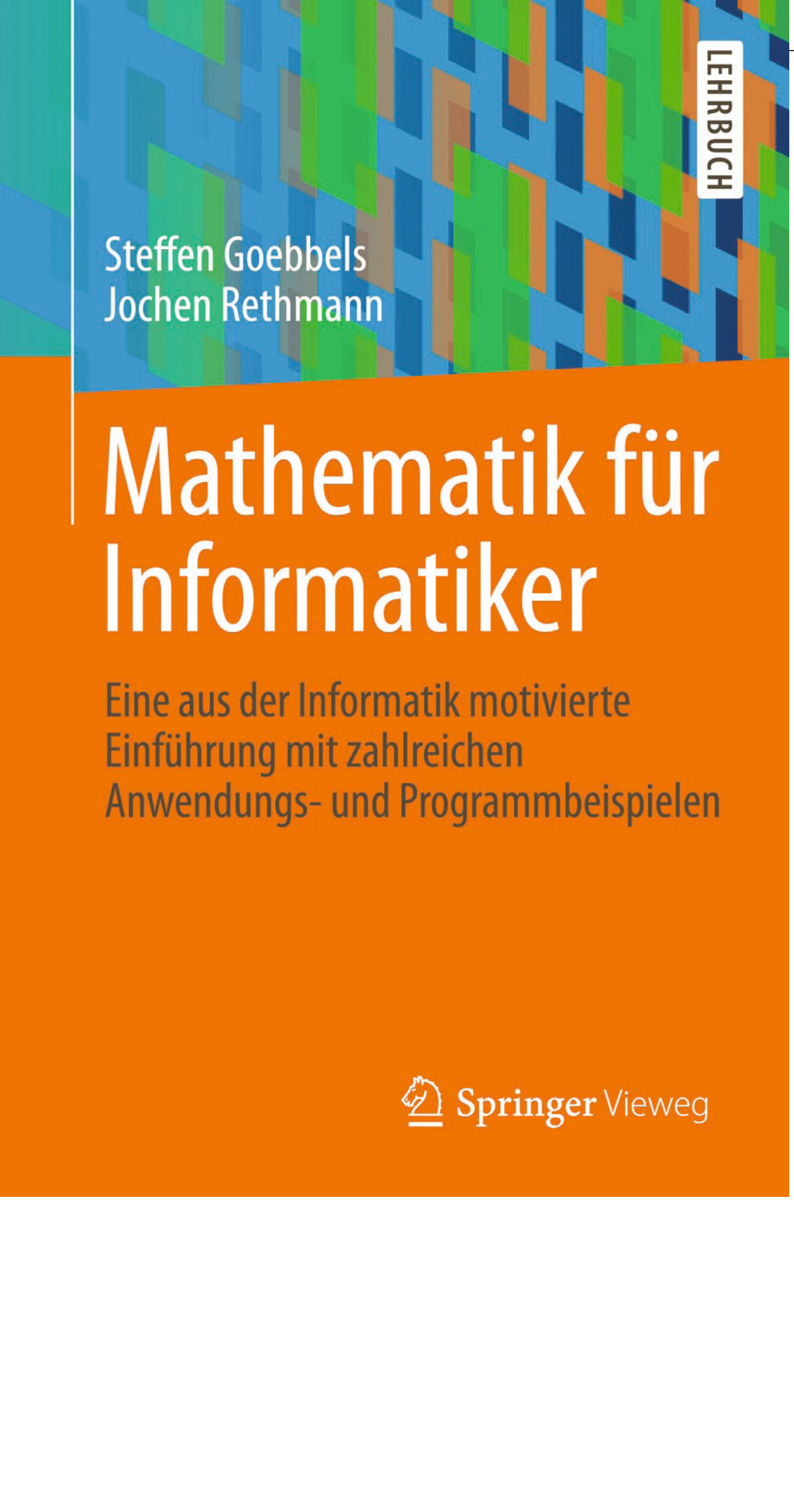
LEHRBUCH
Steffen Goebbels
Jochen Rethmann
Mathematik für Informatiker
Eine aus der Informatik motivierte Einführung mit zahlreichen Anwendungs- und Programmbeispielen
Springer Vieweg

Steffen Goebbels • Jochen Rethmann

Mathematik für Informatiker

Eine aus der Informatik motivierte Einführung mit zahlreichen Anwendungs- und Programmbeispielen

Springer Vieweg

Steffen Goebbels
Jochen Rethmann

Fachbereich Elektrotechnik und Informatik
Hochschule Niederrhein
Krefeld, Deutschland

ISBN 978-3-642-55339-4 ISBN 978-3-642-55340-0 (eBook)
DOI 10.1007/978-3-642-55340-0

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

Springer Vieweg

Planung und Lektorat: Dr. Andreas Rüdinger, Barbara Lühker

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE.
Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media.
www.springer-vieweg.de

Vorwort

Wenn Sie gerade in den ersten Semestern eines Informatik-Studiums sind und sich fragen, warum Sie so viel Mathe lernen müssen, obwohl Sie doch Informatik studieren, dann halten Sie genau das richtige Buch in den Händen.

Ihre Frage ist sehr berechtigt, denn in der Software-Entwicklung, beim Design von Mensch-Maschine-Schnittstellen, bei Betriebssystemen, beim Planen und Betreiben von Computer-Netzwerken, beim Compilerbau, bei Programmiersprachen oder bei rechtlichen und gesellschaftlichen Aspekten der Informatik benötigt man so gut wie keine Mathematik. Das Argument, Mathematik schult das abstrakte Denken, ist zu fadenscheinig, als dass es wirklich Motivation wäre. Motivation ist aber für ein erfolgreiches Lernen extrem wichtig. Wir wollen daher in diesem Buch genau in die Mathematik einführen, die in den Kernfächern der Informatik benötigt wird. Die Gesellschaft für Informatik listet in Ihren Empfehlungen für Informatik-Studiengänge folgende Module auf, die auf jeden Fall unterrichtet werden sollen: Theoretische Informatik, Algorithmen und Datenstrukturen, Rechnerstrukturen, Rechnernetze, Webbasierte Anwendungen, Betriebssysteme, Verteilte Systeme, Datenbanken und Informationssysteme, Wissensbasierte Systeme sowie Sicherheit und Sicherheitstechniken. Wir haben uns diese Module angesehen und die wesentlichen Teile der dafür benötigten Mathematik in diesem Buch zusammengestellt.

Andererseits ist die Informatik ein Dienstleistungsfach, das für andere Disziplinen Software entwickelt. Die Probleme und Fragestellungen kommen aus anderen Wissensgebieten, die Lösungen und Programme aus der Informatik: Bio-, Chemo-, Geo-, Medizin-, Medien-, Sozio- oder Wirtschaftsinformatik. Es ist nicht Ziel dieses Buches, sämtliche Mathematik für alle Wissensgebiete bereit zu stellen. Dann müsste man die Mathematik in der vollen Breite und Tiefe behandeln, da jedes dieser Wissensgebiete andere Teile der Mathematik benötigt.

Wir beginnen mit der Bool'schen-Algebra, mit der wir ein Addierwerk bauen. Damit eng verwandt sind Aussagen- und Prädikatenlogik, die in ihrer elementaren Form beim Programmieren benötigt werden, aber auch die Grundlage für Programmiersprachen wie Prolog sind. Dann sehen wir uns Abbildungen (Funktionen) an, die wir mittels Relationen einführen. Das Konzept der Relationen ist die Grundlage für die heute überwiegend eingesetzten relationalen Datenbanken. Wie die unterschiedlichsten Probleme mittels mathematischer Modelle gelöst werden können, sehen wir dann im Kapitel Graphentheorie. Viel Platz nimmt auch die Betrachtung der Zahlen, ihrer Rechenregeln (Algebra) und ihrer Darstellung im Computer ein. Die Absicherung von WLAN, die Authentifizierung der Server beim Client sowie die Verschlüsselung der Daten bei der Übertragung vom Web-Browser zu einem Bank-Server sind Beispiele, in denen Primzahlen für die Verschlüsselung eingesetzt werden. Wir nehmen das zum

Anlass, um zu zeigen, wie schwere Probleme wie Primzahltests, die nur mit lang laufenden Programmen gelöst werden können, sich mittels Randomisierung oft enorm beschleunigen lassen. Dazu machen wir einen Exkurs in die Wahrscheinlichkeitsrechnung. Für weitere Laufzeitabschätzungen, die wir für einige Sortierverfahren durchführen, sehen wir uns Folgen und Reihen an. Zum Vergleich von Laufzeiten nutzen wir Grenzwerte, bei deren Berechnung uns die Differenzial- und Integralrechnung hilft. Über Codes und speziell lineare Codes motivieren wir im letzten Kapitel die wesentlichen Ergebnisse der Linearen Algebra, also der Vektorrechnung. Dazu gehören insbesondere Sätze über lineare Gleichungssysteme, die elementarer Bestandteil der meisten mathematischen Modelle sind. Für die Computer-Grafik betrachten wir schließlich, wie die Lineare Algebra zum Drehen und Verschieben von Objekten benutzt werden kann.

Wir setzen die Mittelstufenmathematik als bekannt voraus. Falls Sie Ihre Erinnerungen (z. B. an die Bruchrechnung) auffrischen möchten, empfehlen wir Gellrich und Gellrich (2003).

Auf der Webseite zum Buch, http://www.springer-spektrum.de/978-3-642-55339-4, finden Sie die Lösungen zu allen gestellten Aufgaben. Fühlen Sie sich ermutigt, selbst über einige Aussagen nachzudenken, bevor Sie unsere Erklärungen lesen. Nur indem Sie selbst „Mathematik machen“, lernen Sie sie richtig. Außerdem werden Sie so Erfolgserlebnisse bekommen. Dann macht Mathematik Spaß!

Dank

Wir möchten unseren Kollegen in Krefeld und Düsseldorf danken, die uns bei der Erstellung des Buchs unterstützt haben. Besonderer Dank gilt Prof. Dr. Christoph Dalitz, Dr. habil. Frank Gurski, Prof. Dr. Regina Pohle-Fröhlich, Prof. Dr. Ulrich Tipp, Prof. Dr. Peer Ueberholz sowie der Stadt Krefeld, die uns Daten für ein Stadtmodell zur Verfügung gestellt hat.

Zum Schluss möchten wir uns noch ganz besonders bei Herrn Dr. Rüdinger und Frau Lühker von Springer-Spektrum bedanken, die das Buchprojekt ermöglicht haben und uns mit professioneller Hilfe zur Seite standen.

Hier verwendete Programmiersprachen

Wir werden einige Programmbeispiele in der Programmiersprache C angeben. Um diese zu verstehen, benötigen Sie ganz elementare Kenntnisse in einer Sprache, die eine C-ähnliche Syntax wie beispielsweise Java, C# oder C++ hat. Wenn Sie noch nicht programmiert haben, dann empfehlen wir die ersten Seiten des auch im Internet kostenlos verfügbaren Buchs Wolf (2009).

Algorithmen sind Kochrezepte, die eindeutig beschreiben, wie ein Problem gelöst werden kann. Solche Algorithmen werden wir halb-formal beschreiben. Dazu benutzen wir eine Art Programmiersprache, sogenannten Pseudo-Code, deren Anweisungen in der Regel wie in C von oben nach unten abgearbeitet werden. Änderungen in dieser Reihenfolge bewirken die folgenden Schlüsselworte:

if Bedingung **then**
    Anweisungen A
else
    Anweisungen B

Ist die Bedingung erfüllt, dann werden die unter dem **if** eingerückten Anweisungen A ausgeführt. Ist die Bedingung nicht erfüllt, dann werden die Anweisungen B abgearbeitet. Die **else**-Anweisung zusammen mit den Anweisungen B ist optional.

for Variable := Start- bis Endwert **do**
    Anweisungen

Die Anweisungen werden in der **for**-Schleife für jeden Wert der Variable vom Startwert bis zum Endwert in dieser Reihenfolge ausgeführt.

for all Element aus Menge **do**
    Anweisungen

Bei dieser **for**-Schleife werden die Anweisungen für alle Elemente einer Menge aufgerufen, wobei die Reihenfolge der Aufrufe nicht festgelegt ist.

while Bedingung **do**
    Anweisungen

repeat
    Anweisungen
until Bedingung

Die Anweisungen werden in diesen Programmfragmenten so lange immer wieder ausgeführt, wie es die Bedingung vorgibt. Bei der kopfgesteuerten **while**-Schleife wird zuerst die Bedingung geprüft. Ist sie erfüllt, werden die Anweisungen ausgeführt. Dagegen werden bei der fußgesteuerten **repeat-until**-Schleife die Anweisungen vor der Prüfung der Bedingung abgearbeitet. Diese Schleife wird im Gegensatz zur **while**-Schleife solange ausgeführt, bis die Bedingung wahr wird.

Wollen wir einen Programmteil als Prozedur (als Funktion) realisieren, die wir mit Parametern aufrufen können und die daraus einen Rückgabewert berechnet, so definieren wir die Prozedur über

procedure PROZEDURNAME(Variable 1, Variable 2, ...)
    Anweisungen
    return Rückgabewert

Nach Ausführung einer **return**-Anweisung ist die Prozedur beendet. Die Prozedur wird im Programm anhand des Namens aufgerufen, wobei in den Klammern konkrete Werte oder Variablen des aufrufenden Programmteils stehen:

Ergebnis := PROZEDURNAME(Wert 1, Wert 2,...)

Wir haben in diesen Programmschnipseln die Zuweisung := verwendet. In der Mathematik ist $x = x + 1$ eine stets falsche Aussage, da auf der linken Seite ein anderer Wert als auf der rechten Seite steht. In Algorithmen sieht man dagegen häufig solche Ausdrücke (z. B. bei C-Programmen). Gemeint ist hier kein Vergleich sondern die Zuweisung „ändere den Wert von x zum Wert $x+1$“, die wir in den halbformalen Algorithmen mit := vom Vergleich unterscheiden.

Inhaltsverzeichnis

1 Grundlagen

Übersicht

1.1 Mengen

Beim Programmieren beschäftigen wir uns einerseits mit Datenstrukturen und andererseits mit Algorithmen, die auf diesen Datenstrukturen arbeiten. In der Mathematik ist das ähnlich. So werden beispielsweise Arrays (zusammenhängende Speicherbereiche) für Zahlen benötigt, die in der Mathematik Vektoren oder Matrizen heißen. In einer Matrix gibt es feste Plätze, denen Zahlen zugeordnet sind. Damit kann die gleiche Zahl an verschiedenen Plätzen stehen und damit mehrfach vorkommen. Außerdem ist durch die Anordnung eine Reihenfolge der Zahlen vorgegeben. Oft sind aber Reihenfolge und Anzahl unwichtig. Eine Struktur, mit der nur „gespeichert“ wird, ob ein Wert vorkommt oder nicht, ist die Menge.

1.1.1 Mengen und Schreibweisen

Definition 1.1 (Mengenbegriff von Cantor)
Eine **Menge** M ist eine gedankliche Zusammenfassung von unterscheidbaren Dingen. Diese Dinge heißen die **Elemente** von M.

Diese Definition ist nicht besonders genau, da wir beispielsweise von „Dingen“ sprechen und nicht erklären, was wir genau darunter verstehen. Häufig sind es Zahlen, aber es können auch ganz andere Objekte sein. Was eine „gedankliche Zusammenfassung“ ist, verraten wir auch nicht, aber Mengen werden in der Regel durch eine in geschweifte Klammern gesetzte Auflistung der Elemente beschrieben. Damit drücken die Klammern „{“ und „}“ die Zusammenfassung

S. Goebbels, J. Rethmann, *Mathematik für Informatiker,*
DOI 10.1007/978-3-642-55340-0_1, © Springer-Verlag Berlin Heidelberg 2014

aus. Die Menge der ganzen Zahlen von eins bis sechs wird also als $\{1, 2, 3, 4, 5, 6\}$ geschrieben. Hat eine Menge kein einziges Element, dann nennen wir sie die **leere Menge** und schreiben $\{\}$ oder $\emptyset$. Vorsicht: Die Menge $\{\emptyset\}$ ist nicht leer! Sie hat ein Element, nämlich die leere Menge.

Wenn wir zweimal hintereinander Würfeln, dann können wir das Ergebnis über eine Menge von **Paaren** darstellen:

$$\{(1,1),(1,2),\ldots,(1,6),(2,1),(2,2),(2,3),\ldots,(6,6)\}.$$

Wenn wir dagegen das Ergebnis als Summe der beiden Würfe notieren, ist es Element von $\{2, 3, \ldots, 12\}$. So werden wir Mengen zur Modellierung in der Wahrscheinlichkeitsrechnung einsetzen.

Die Menge der Quadratzahlen ist

$$\{1, 4, 9, 16, 25, \ldots\} = \{n^2 : n \in \{1, 2, 3, \ldots\}\},$$

die Menge der Zweierpotenzen lautet

$$\{1, 2, 4, 8, 16, 32, \ldots\} = \{2^n : n \in \{0, 1, 2, 3, \ldots\}\}.$$

Hier haben wir eine weitere Schreibweise verwendet: Wir beschreiben die Elemente mit n^2 und 2^n, wobei wir dann n weiter spezifizieren. Der Doppelpunkt, für den häufig auch ein senkrechter Strich verwendet wird, wird als „wofür gilt“ gelesen. Nun soll n null oder eine natürliche Zahl sein, also in der Menge $\mathbb{N}_0 := \{0, 1, 2, 3, \ldots\}$ enthalten sein. Das Zeichen „$\in$“ bedeutet „Element von“. Damit ist die Menge der Quadratzahlen die Menge aller der Zahlen n^2, wofür n Element von $\mathbb{N} := \{1, 2, 3, \ldots\}$ gilt.

Die Mengen der Quadratzahlen und der Zweierpotenzen werden über die bereits bekannte Menge der natürlichen Zahlen definiert. Man bildet gerne neue Mengen mit der Hilfe von bereits definierten Mengen. So kann man auch Mengen mit unendlich vielen Elementen gut beschreiben. Außerdem gibt es mit diesen Mengen keine Existenzprobleme. Anders sieht es beispielsweise bei der Menge aus, die aus allen Mengen besteht. Es lässt sich zeigen, dass diese Allmenge nicht existieren kann, da sonst Widersprüche entstehen (siehe Aufgabe 1.9 auf Seite 26). Definition 1.1 verbietet die Allmenge aber nicht. Daher spricht man von einem naiven Mengenbegriff. Für unsere Zwecke reicht er aber völlig aus. Eine genauere Definition, die auf Axiomen beruht, können Sie aber in (Ebbinghaus, 2003, Kapitel III) nachlesen.

Definition 1.2 (Mengenschreibweisen)

- So wie wir bereits $u \in V$ geschrieben haben, um auszudrücken, dass u Element der Menge V ist, schreiben wir $u \notin V$, falls u kein Element von V ist.

- Zwei Mengen U und V heißen **gleich** genau dann, wenn sie die gleichen Elemente besitzen, d. h., jedes Element der einen Menge findet sich in der anderen und umgekehrt. Wir schreiben $U = V$. Gilt dies nicht, so schreiben wir $U \neq V$. Dann besitzt eine der Mengen ein Element, das kein Element der anderen ist.
  Wegen dieser Definition der Gleichheit spielt die Reihenfolge, mit der wir Elemente einer Menge auflisten, keine Rolle. Auch prüft man nur, ob ein Element vorkommt – nicht wie oft es vorkommt, da die Elemente unterscheidbar sind und damit nur einmal vorkommen können. Damit eignet sich der klassische Mengenbegriff nicht, das mehrfache Enthaltensein eines Elements zu modellieren. Unter der Anzahl der Elemente einer Menge (auch **Mächtigkeit** der Menge) verstehen wir also die Anzahl der verschiedenen Elemente.
- U heißt **Teilmenge** von V genau dann, wenn jedes Element von U auch Element von V ist. Wir schreiben $U \subset V$. Offensichtlich ist $\emptyset \subset V$, aber auch $V \subset V$ ist richtig. Einige Autoren verwenden das Symbol „$\subseteq$“ um explizit auszudrücken, dass Gleichheit gestattet ist. Sie benutzen dann „$\subset$“ nur, wenn U eine **echte Teilmenge** von V ist, d. h. wenn U Teilmenge von V ist, es in V aber mindestens ein Element gibt, das in U nicht enthalten ist. Dafür finden Sie allerdings auch die Schreibweise $U \subsetneq V$.
  Ist U keine Teilmenge von V, so schreiben wir $U \not\subset V$.
- Wir können nun zu einer gegebenen Menge V alle Teilmengen betrachten und diese als Elemente einer neuen Menge auffassen. Diese Menge aller Teilmengen von V heißt die **Potenzmenge** $\mathcal{P}(V)$ von V. $\mathcal{P}(V)$ ist eine Menge von Mengen. Offensichtlich ist $\emptyset \in \mathcal{P}(V)$ und $V \in \mathcal{P}(V)$.
- Das Komplement einer Menge V ist die Menge mit den Elementen, die in V nicht enthalten sind. Diese Formulierung hat ein Problem: Wir benötigen eine Grundmenge G, aus der wir diese Elemente nehmen können. Sei also $V \subset G$. Dann ist das **Komplement** $\mathcal{C}_G V$ der Menge V hinsichtlich der Grundmenge G definiert über

  $$\mathcal{C}_G V := \{u \in G : u \notin V\}.$$

  Wenn aus dem Zusammenhang die Grundmenge G bekannt ist, dann können wir sie in der Notation auch weglassen. Häufig findet man daher die Schreibweisen

  $$\overline{V} := \mathcal{C}V := \mathcal{C}_G V.$$

- Mengen U und V heißen **disjunkt** oder **elementfremd** genau dann, wenn sie keine gemeinsamen Elemente besitzen.

- Möchten wir besondere Elemente aus einer Menge „auslesen“, so können wir der Menge einen entsprechenden Operator voranstellen. Sind die Elemente beispielsweise der Größe nach vergleichbar und unterschiedlich groß, so liefert $\max V$ das größte Element, falls es existiert. Es existiert in jedem Fall, wenn die Menge nur endlich viele Elemente hat. Entsprechend bezeichnet $\min V$ das kleinstes Element.

Dass wir bei Potenzmengen Mengen selbst wieder als Elemente einer Menge auffassen, erscheint merkwürdig. Betrachten wir eine Firma, die n Produkte herstellt (Menge mit n Elementen) und nun Bündel aus k Produkten für eine Anzahl $0 < k \leq n$ verkaufen möchte: Jedes Bündel ist eine Teilmenge mit k Elementen (später werden wir solche Teilmengen „Kombinationen von k aus n Elementen“ nennen, siehe Kapitel 3.2.4). Jedes Bündel ist also eine Teilmenge der Menge der Produkte. Die Menge aller Bündelprodukte ist eine Teilmenge der Potenzmenge. Einzig die leere Menge als Element der Potenzmenge wäre kein sinnvolles Bündelprodukt (sofern die Kunden nicht betrogen werden sollen).

Beispiel 1.3
Im Kapitel 2 beschäftigen wir uns intensiv mit Graphen. Diese bestehen aus Knoten, die z. B. durch gerichtete Kanten verbunden sein können. Ist V die Menge der Knoten, so können wir eine gerichtete Kante von einem Knoten u zu einem Knoten v über ein Paar (u, v) darstellen. Die Menge aller möglichen Kanten werden wir über $\{(u, v) : u, v \in V, u \neq v\}$ definieren, wobei wir wegen $u \neq v$ (u ungleich v) keine Kanten von einem Knoten zu sich selbst zulassen.

Wir betrachten einen konkreten Graphen mit Knoten $V := \{a, b, c\}$ und gerichteten Kanten $E := \{(a, b), (b, a), (c, a)\}$. In Abbildung 1.1 sind diese Kanten als Pfeile gezeichnet.

- Das Element (b, a) ist eine Kante, aber (b, c) ist keine Kante, denn $(b, a) \in E$ und $(b, c) \notin E$.
- Es ist $E \subset \{(u, v) : u, v \in V, u \neq v\} = \{(a, b), (a, c), (b, a), (b, c), (c, a), (c, b)\}$.
- Das Komplement von E bezüglich der Menge aller möglichen Kanten $G := \{(u, v) : u, v \in V, u \neq v\}$ ist die Menge $\mathcal{C}_G E = \{(a, c), (b, c), (c, b)\}$.
- $\mathcal{P}(V) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, V\}$.

■

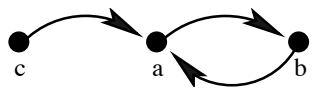

Abb. 1.1 Graph zu Beispiel 1.3

Satz 1.4 (Potenzmenge)
Eine Menge V mit n Elementen besitzt 2^n verschiedene Teilmengen, also hat die zugehörige Potenzmenge 2^n Elemente.

Beweis Wie häufig in der Informatik benutzen wir einen Entscheidungsbaum. Beim Bilden einer Teilmenge können wir für jedes Element von V entscheiden, ob wir es in die Teilmenge aufnehmen oder nicht. Eine abweichende Entscheidung führt zu einer anderen Teilmenge. Die n Entscheidungen führen damit zu den 2^n verschiedenen Teilmengen, siehe dazu den Binärbaum in Abbildung 1.2. (Bei einem Binärbaum folgen auf jeden Knoten, hier als Kästchen gezeichnet, maximal zwei nachgelagerte Knoten. Wir betrachten Binärbäume auf Seite 33 genauer.) □

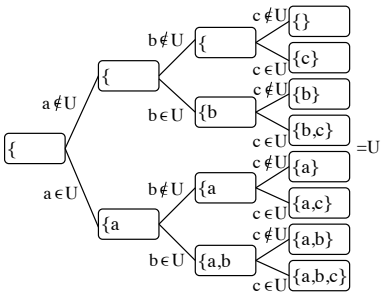

Abb. 1.2 Entscheidungsbaum zum Auffinden aller 2^3 Teilmengen U von $V := \{a, b, c\}$

Der Überlieferung nach durfte der Erfinder des Schachspiels als Belohnung für seine Erfindung einen Wunsch an seinen König äußern. Er wünschte sich Weizenkörner - und zwar so viele, wie sich ergeben, wenn man auf das erste Schachfeld ein Korn legt und dann von Feld zu Feld die Anzahl verdoppelt. Das sind insgesamt $1 + 2 + 4 + 8 + \cdots + 2^{63}$ Körner. Der König hielt diesen Wunsch zunächst für bescheiden, aber dann kam das böse Erwachen. Denn wenn wir die Zahlen addieren, erhalten wir $2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$. Die Zahl $2^{64} - 1$ ergibt sich übrigens über eine Formel für die geometrische Summe, siehe Seite 34. Was bedeutet nun diese Zahl? Wir treffen die Annahme, dass ein Korn etwa 0,05 Gramm wiegt (das ist unsere Schätzung, wir sind dafür aber keine Experten). Alle $2^{64} - 1$ Weizenkörner wiegen dann zusammen 922 337 203 685 477 kg. Ein ISO-Container, der üblicherweise in der Handelsschifffahrt eingesetzt wird, hat die Maße 12,192 m ×2,438 m ×2,591 m und ein maximales Ladegewicht von 26 630 kg. Wir benötigen also 34 635 268 632 Container, die zusammen eine Länge von 416 731 552 187 Metern haben. Das Licht legt in einer Sekunde etwa 300 000 km zurück. Reihen wir also die Container aneinander, so benötigt das Licht für diese Strecke 1389 Sekunden, oder anders gesagt: mehr als 23 Minu-

ten. Das Licht benötigt für die Strecke von der Sonne zur Erde nicht einmal 9 Minuten!

Wir sehen also, dass 2^n sehr schnell mit n anwächst und für $n = 64$ für praktische Zwecke bereits viel zu groß ist. Leider führen viele Probleme in der Informatik bei naiver Lösung zu Programmlaufzeiten, die sich in Abhängigkeit der Eingabegröße n des Problems wie 2^n verhalten. Hier müssen wir dann nach geschickteren Lösungen suchen.

1.1.2 Mengenoperationen

Wir benötigen Mengen zur Beschreibung von komplizierten realen Problemen durch Modelle. Beispielsweise werden Ereignisse, deren Eintrittswahrscheinlichkeit berechnet werden soll, über Mengen ausgedrückt (siehe Kapitel 3.3.1). Dabei ist es nötig, gewisse Operationen auf Mengen durchführen zu können – ähnlich wie wir es von den Zahlen kennen.

Definition 1.5 (Mengenoperationen)
Es seien U und V Mengen.

- Die **Schnittmenge** von U und V ist die größte gemeinsame Teilmenge von U und V, d. h., sie enthält alle Elemente von U, die auch in V enthalten sind:
$$U \cap V := \{u \in U : \ u \in V\}.$$
Offensichtlich gilt: $U \cap V = V \cap U$. Wir sprechen $U \cap V$ als „U geschnitten mit V“.
- Die **Vereinigungsmenge** von U und V ist die kleinste Menge, die U und V als Teilmengen hat:
$$U \cup V := \{u : \ u \in U \text{ oder } u \in V\}.$$
Auch das Vereinigen von Mengen ist **kommutativ** (vertauschbar): $U \cup V = V \cup U$, und wir sprechen $U \cup V$ als „U vereinigt mit V“.
- Wenn wir eine Menge V von einer Menge U subtrahieren, dann lassen wir in U die Elemente weg, die in V enthalten sind. Genauer ist die **Differenz** von U und V definiert über
$$U \setminus V := \{u \in U : \ u \notin V\}.$$
Wir sprechen $U \setminus V$ als „U ohne V“. V darf auch Elemente besitzen, die keine Elemente von U sind.

- Um **Paare** (u, v) zu bilden, deren erster Eintrag aus einer Menge U und deren zweiter Eintrag aus einer Menge V stammt, verwenden wir das **Kreuzprodukt** (das **kartesische Produkt**) von U und V:

$$U \times V := \{(u, v) : \ u \in U \text{ und } v \in V\}.$$

  Entsprechend können wir **n-Tupel** bilden (Paare sind 2-Tupel). Bei Relationen und in der Linearen Algebra benötigen wir den Fall, dass alle Einträge eines n-Tupels aus der gleichen Menge stammen,

$$U^n := \underbrace{U \times U \times \cdots \times U}_{n\text{-mal } U} = \{(u_1, u_2, \ldots, u_n) : \ u_1, u_2, \ldots, u_n \in U\}.$$

Die eingangs beschriebene Modellierung eines zweimaligen Würfelns ist ein Kreuzprodukt:

$$\{(1,1),(1,2),\ldots,(1,6),(2,1),\ldots,(6,6)\} = \{1,\ldots,6\} \times \{1,\ldots,6\}.$$

In Beispiel 1.3 sind die Kanten Elemente der Menge $V^2 = V \times V$. Kombiniert man Mengenoperationen, so ist es oft lästig, mit Klammern die Abarbeitungsreihenfolge vorzugeben. Daher wird festgelegt, dass das Komplement am engsten bindet, dann folgen Durchschnitt und Vereinigung, also ist beispielsweise $\mathcal{C}U \cup V \cap W = (\mathcal{C}U) \cup (V \cap W)$. Das ist vergleichbar mit den Punkt-vor-Strich-Regeln beim Rechnen mit Zahlen: $3 + 4 \cdot 5 = 3 + (4 \cdot 5)$.

Beispiel 1.6 (Mengenverknüpfungen)
Seien M_1 und M_2 die Mengen der Methoden zweier Klassen eines objektorientierten Programms:

$$M_1 := \{\text{getName}, \text{getAddress}, \text{setName}, \text{setAddress}, \text{getChangeDate}\}$$
$$M_2 := \{\text{getName}, \text{getAddress}, \text{print}\}$$

Wir könnten gemeinsame Methoden in eine Basisklasse verlagern:

$$M_1 \cap M_2 = \{\text{getName}, \text{getAddress}\}.$$

Dann wären gegebenenfalls alle Methoden

$$M_1 \cup M_2 = \{\text{getName}, \text{getAddress}, \text{setName}, \text{setAddress}, \text{getChangeDate}, \text{print}\}$$

nur einmal zu implementieren. Speziell für M_2 muss nur

$$M_2 \setminus M_1 = \{\text{print}\}$$

programmiert werden. ■

Komplement und Differenz von Mengen führen häufig zu Irritationen. Beim Komplement $\mathcal{C}_G U$ benötigen wir zu einer Menge U eine Grundmenge, die U enthält: $U \subset G$. Bei der Mengendifferenz können die beteiligten Mengen beliebig zueinander liegen. Allerdings lässt sich das Komplement als Mengendifferenz schreiben. Seien $U \subset G$ und $V \subset G$, dann gilt:

$$U \setminus V = U \cap \mathcal{C}_G V, \quad \mathcal{C}_G U = G \setminus U.$$

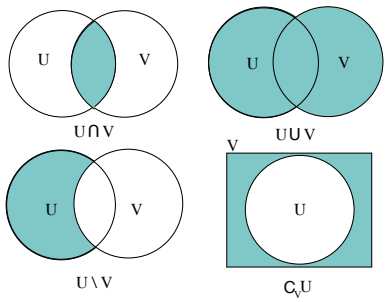

Abb. 1.3 Mengenoperationen, dargestellt als Venn-Diagramme

Um zu veranschaulichen, was bei den Mengenoperationen geschieht, ist es üblich, Mengen durch Flächen zu visualisieren. Streng genommen betrachtet man dabei nur Teilmengen von Punkten der Zeichenebene, die aber beliebige Mengen repräsentieren sollen. Die Schnittmenge zweier Mengen ist dann die Fläche, auf der sich die Mengen überlappen. Bei der Vereinigung fügt man Flächen zu einer gemeinsamen zusammen. Diese Darstellungsart nennt man **Venn-Diagramm**. Dabei ist es üblich, die Mengen als Kreisflächen, Ovale oder Rechtecke zu malen, siehe Abbildung 1.3. Mit Venn-Diagrammen können Sie leicht die Gültigkeit der folgenden Regeln nachprüfen:

Satz 1.7 (Eigenschaften von Mengen)
Es seien U, V und W Mengen. Dann gilt:

- *Aus $U \subset V \subset W$ folgt $U \subset W$, d. h., die Teilmengenbeziehung ist* **transitiv**.
- *Ist $U \subset V$ und gleichzeitig $V \subset U$, dann folgt $U = V$.*
- **Kommutativgesetze** *(Vertauschung der Reihenfolge, s. o.):*

  $$U \cup V = V \cup U, \qquad U \cap V = V \cap U.$$

  Im Gegensatz dazu spielt aber z. B. bei der Mengendifferenz die Reihenfolge durchaus eine Rolle.
- **Assoziativgesetze***: Bei gleichartigen Operationen kann die Reihenfolge beliebig gewählt werden, d. h., die Klammerung ist unwichtig:*

  $$U \cup (V \cup W) = (U \cup V) \cup W, \qquad U \cap (V \cap W) = (U \cap V) \cap W.$$

- **Distributivgesetze**: *Analog zur Ausmultiplikation* $x \cdot (y+z) = x \cdot y + x \cdot z$ *für Zahlen* x, y *und* z *gilt:*

  $$U \cap (V \cup W) = (U \cap V) \cup (U \cap W), \quad U \cup (V \cap W) = (U \cup V) \cap (U \cup W).$$

  Die Regeln gelten also auch bei Vertauschung von Vereinigung und Durchschnitt. Das ist beim Distributivgesetz für Zahlen nicht der Fall: In der Regel ist $x + (y \cdot z) \neq (x + y) \cdot (x + z)$, *wie wir am Beispiel* $4 + (2 \cdot 5) \neq (4 + 2) \cdot (4 + 5)$ *sehen.*
- *Bei der Bildung des Komplements sind die* **De Morgan'schen Regeln** *zu beachten. Für* $U, V \subset W$ *gilt:*

  $$\mathcal{C}_W(U \cup V) = \mathcal{C}_W U \cap \mathcal{C}_W V, \quad \mathcal{C}_W(U \cap V) = \mathcal{C}_W U \cup \mathcal{C}_W V.$$

  Unter der Komplementbildung kehrt sich das Operatorsymbol um. Wir werden analoge Regeln gleich anschließend für die Digitaltechnik und Logik kennen lernen.

Aufgabe 1.1
Gegeben sind die Mengen $U := \{1, 3, 5\}$, $V := \{1, 2, 4, 5, 7, 8, 9, 10\}$ *und* $G := \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

a) *Geben Sie alle Teilmengen von* U *an.*
b) *Berechnen Sie:* $U \cup V$, $U \cap V$, $U \setminus V$, $V \setminus U$, $\mathcal{C}_G(U \cap V)$, $(\mathcal{C}_G U) \setminus V$ *sowie* $\mathcal{C}_G[(\mathcal{C}_G U) \cap V] \cup U$.

Diese Aufgabe ist vergleichsweise leicht. Sie werden im Folgenden Aufgaben vorfinden, die einen unterschiedlichen Schwierigkeitsgrad besitzen. Diese sollten von Ihnen nicht in der Art gelöst werden, dass Sie die Internet-Seite zum Buch aufsuchen und sich die Lösung dort ansehen. Wir verstehen unter einer Übungsaufgabe, dass Sie selbst Ihren Verstand zum Lösen einsetzen sollen. Das ist zwar mühsamer, aber auch sehr viel lehrreicher und befriedigender.

1.2 Logik

1.2.1 Rechnen mit Nullen und Einsen: Boole'sche Algebra

In digitalen Computern gibt es nur die elementaren Zustände null und eins. Was sich mit diesen **Bits** alles codieren lässt, beschreiben wir in späteren Kapiteln. Hier beschränken wir uns zunächst auf die beiden elementaren Zustände. Es gibt 16 Möglichkeiten, Verknüpfungen von zwei Bits (als Werte der Varia-

blen A und B) zu definieren. In der Tabelle 1.1 werden diese 16 verschiedenen Verknüpfungen als Spalten 0 bis 15 dargestellt.

Tab. 1.1 Mögliche Verknüpfung zweier Bits

A	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Einige dieser Verknüpfungen haben einen Namen und werden durch spezielle Symbole beschrieben:

- Spalte 1: Die Werte von A und B werden und-verknüpft. Nur wenn beide Werte 1 sind, ist auch das Ergebnis 1. Wir schreiben dafür $A \wedge B$ (A und B) oder auch $A \odot B$ um anzudeuten, dass hier zwei Bits multipliziert werden.
- Spalte 6: Die Werte von A und B werden entweder-oder-verknüpft. Nur wenn genau ein Wert 1 ist, ist auch das Ergebnis 1. Wir schreiben dieses exklusive Oder mit $A \operatorname{xor} B$ bzw. $A \oplus B$. Die Schreibweise mit $\oplus$ deutet an, dass zwei Bits ohne Berücksichtigung eines Übertrags addiert werden.
- Spalte 7: Die Werte von A und B werden oder-verknüpft. Genau dann, wenn mindestens ein Wert 1 ist, ist auch das Ergebnis 1. Wir schreiben $A \vee B$ (A oder B). Im Gegensatz zum exklusiven Oder ist das Ergebnis auch dann 1, wenn sowohl A als auch B gleich 1 sind. In der Umgangssprache wird häufig nicht sauber zwischen „oder“ und „entweder oder“ unterschieden. Hier müssen wir genauer sein.
- Spalte 9: Hier steht das Ergebnis der Verknüpfung $A \iff B$, das genau dann 1 ist, wenn entweder A und B beide 1 oder A und B beide 0 sind. Diese **Äquivalenz** von A und B liefert also genau dann eine 1, wenn A und B den gleichen Wert haben. Auf die Bedeutung der Äquivalenz beim Aufschreiben von Mathematik gehen wir später ein, hier ist sie erst einmal ein Bit-Operator.
- Spalte 12: Die Werte sind genau umgekehrt zu denen von A. Hier ist die Negation von A beschrieben, die mit $\neg A$ bezeichnet wird. Damit können wir die Äquivalenz auch so ausdrücken:

  $$(A \iff B) = (A \wedge B) \vee (\neg A \wedge \neg B).$$

  Das Gleichheitszeichen bedeutet, dass für alle vier Wertkombinationen von A und B beide Seiten das gleiche Ergebnis liefern. Die Formel

  $$(A \iff B) \iff [(A \wedge B) \vee (\neg A \wedge \neg B)]$$

würde also für alle Wertbelegungen von A und B stets die 1 berechnen. Für das exklusive Oder gilt:

$$A \oplus B = (\neg A \wedge B) \vee (A \wedge \neg B).$$

- Spalte 13: Hier steht das Ergebnis der Verknüpfung $A \Longrightarrow B$, das genau dann 1 ist, wenn B gleich 1 oder A gleich 0 ist. Dies ist die **Folgerung** von B aus A, also
$$(A \Longrightarrow B) = (\neg A \vee B).$$
Den Einsatz von Äquivalenzen und Folgerungen werden wir uns später noch wesentlich genauer ansehen.

Die Ergebnisse der anderen Spalten können wir jetzt ebenfalls mit diesen Begriffen ausdrücken, siehe Tabelle 1.2.

Tab. 1.2 Schreibweisen für die Bit-Verknüpfungen

A	B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	A $\wedge$ B	A $\wedge$ $\neg B$	A	$\neg A$ $\wedge$ B	B	A $\oplus$ B	A $\vee$ B	$\neg A$ $\wedge$ $\neg B$	A $\Longleftrightarrow$ B	$\neg B$	A $\vee$ $\neg B$	$\neg A$	A $\Longrightarrow$ B	$\neg A$ $\vee$ $\neg B$	1

In der Programmiersprache C stehen diese bitweisen Verknüpfungen zur Verfügung: Der Und-Operator $\wedge$ heißt hier `&`, statt $\vee$ wird `|` verwendet, und $\oplus$ wird als `^` geschrieben. Dabei werden Worte, die aus mehreren Bits bestehen, positionsweise wie angegeben verknüpft.

- Ausmaskieren: Möchte man gezielt einzelne Bits auf null setzen, dann und-verknüpft man eine Bitfolge mit einer Folge von Einsen, bei der nur die betroffenen Bits null sind. Wir löschen beispielsweise das führende Bit in 101011 durch Verknüpfen mit 011111, also mittels 101011 `&` 011111 = 001011. Wenn man testen möchte, ob ein spezielles Bit gesetzt ist, dann kann man alle anderen ausmaskieren und das Ergebnis mit der Null vergleichen.
- Auffüllen: Mit der oder-Verknüpfung können gezielt einzelne Bits eingeschaltet werden. Die unteren (hinteren bzw. rechten) vier Bits von 101010 werden auf 1 gesetzt mittels 101010 `|` 001111 = 101111.
- Codieren: Die Xor-Verknüpfung lässt sich rückgängig machen, indem man sie nochmal mit der gleichen Bitfolge anwendet:

$$(A \oplus B) \oplus B = A.$$

Auf diese Weise erhält man eine hervorragende Verschlüsselung. Dazu muss man ein Referenzdokument als Schlüssel auf sicherem Weg austauschen, z. B. mit dem RSA-Verfahren, siehe Kapitel 3.5. Statt eines vollständigen Dokuments genügt auch eine Startzahl für einen Zufallszahlengenerator, der basierend auf dieser Zahl ein Dokument (reproduzierbar) generiert. Dann kann man andere Dokumente mit diesem bitweise Xor-verknüpfen um sie zu ver- und entschlüsseln.

Aufgabe 1.2
Zeigen Sie mit einer Wertetabelle, dass $(A \oplus B) \oplus B = A$ *gilt.*

Satz 1.8 (Rechenregeln)
Seien A, B *und* C *Variablen, die die Werte* 0 *und* 1 *annehmen können:*

- **Kommutativgesetze***:*

$$A \wedge B = B \wedge A, \quad A \vee B = B \vee A$$

- **Assoziativgesetze***:*

$$(A \wedge B) \wedge C = A \wedge (B \wedge C), \quad (A \vee B) \vee C = A \vee (B \vee C)$$

- **Distributivgesetze***:*

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C), \quad A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C).$$

- *Völlig analog zum Rechnen mit Mengen gelten auch hier die* **De Morgan'schen Regeln***:*

$$\neg(A \wedge B) = \neg A \vee \neg B, \quad \neg(A \vee B) = \neg A \wedge \neg B.$$

Der Beweis des Satzes ist ganz einfach, man muss nur alle Werte für A, B und C durchprobieren und dabei die jeweilige Formel überprüfen. Das kann man mit einer Wertetabelle wie Tabelle 1.2 tun.

Wir haben beispielsweise $\neg A \wedge \neg B$ geschrieben und meinen damit $(\neg A) \wedge (\neg B)$. Auf die Klammern verzichten wir aber, weil die Operatoren (wie zuvor bei Mengen) Prioritäten haben: Die Negation bindet enger als Und, und Und bindet enger als Oder. Es gilt also:

$$A \wedge \neg B \vee C \wedge D = (A \wedge (\neg B)) \vee (C \wedge D).$$

Die Verknüpfungen $A \wedge B$, $A \vee B$, $\neg A$, $\neg(A \wedge B)$, $\neg(A \vee B)$, $A \oplus B$, usw. lassen sich vergleichsweise einfach mittels Hardware als sogenannte Gatter realisieren. Mit solchen Gattern kann man dann kompliziertere Aufgaben

bewältigen. Als Beispiel machen wir einen Vorgriff auf das übernächste Kapitel und schreiben Zahlen im **Dualsystem** (Zweiersystem). Es ist ein Verdienst des deutschen Computer-Pioniers Konrad Zuse, dass er die erste programmierbare Rechenmaschine konzipiert hat, die auf dem Dualsystem basierte. Dabei gibt es nur die Ziffern 0 und 1; statt Zehnerpotenzen werden Potenzen von 2 verwendet. Die Zahl 10110011 im Dualsystem entspricht der Dezimalzahl $1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 179$. Solchen Zahlendarstellungen widmen wir uns intensiv in Kapitel 3.

Die Addition einstelliger Dualzahlen ist recht einfach. Nur im Fall $1+1$ muss ein Übertrag an die nächst höhere Stelle berücksichtigt werden.

$$\begin{array}{rrrr} 0 & 0 & 1 & 1 \\ +\,0 & +\,1 & +\,0 & +\,1 \\ \hline 0 & 1 & 1 & \mathbf{1}0 \end{array}$$

Mehrstellige Zahlen können, wie wir es aus der Schule kennen, ziffernweise addiert werden. Dabei müssen Überträge an die nächst höhere Stelle berücksichtigt werden. Stellen, an denen ein Übertrag entsteht, sind unterstrichen:

$$\begin{array}{lrlr} \text{Dezimal:} & 37 & \text{Dual:} & 00100101 \\ & +49 & & +00110001 \\ & {\scriptstyle 1} & & {\scriptstyle 1\quad 1} \\ \hline & 8\underline{6} & & 0\underline{1}0101\underline{1}\underline{0} \end{array}$$

Tab. 1.3 Wertetabelle eines Volladdierers

A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C_{in}	0	1	0	1	0	1	0	1
S	0	1	1	0	1	0	0	1
C_{out}	0	0	0	1	0	1	1	1

Wir stellen nun Formeln für die Summe zweier Ziffern A und B mit einem Übertrag C_{in} auf. Bei dieser Addition entsteht ein Summenbit S sowie ein neuer Übertrag C_{out} für die nächste Stelle. Wir erhalten die Formel für S als **disjunktive Normalform**, indem wir in der Wertetabelle 1.3 genau die Spalten betrachten, in denen $S = 1$ ist. Zu jeder Spalte erstellen wir einen Term, der für die hier vorliegende Eingabe A, B und C_{in} die Eins liefert. Ist die Eingabe einer dieser Variablen null, so verwenden wir die negierte Variable, sonst die nicht-negierte. Diese **Literale** (d. h. negierte und nicht-negierte Variablen) werden dann mit Und verknüpft. Das Ergebnis ist eine Formel, die genau bei der

vorliegenden Eingabekonstellation eine Eins erzeugt und sonst stets den Wert null hat. So erhalten wir für S die Formeln

$$\neg A \wedge \neg B \wedge C_{\text{in}}, \quad \neg A \wedge B \wedge \neg C_{\text{in}}, \quad A \wedge \neg B \wedge \neg C_{\text{in}}, \quad A \wedge B \wedge C_{\text{in}}.$$

Wir müssen diese Terme nun lediglich oder-verknüpfen um S zu berechnen. Völlig analog verfahren wir für C_{out} und erhalten zusammen

$$\begin{aligned} S &= (\neg A \wedge \neg B \wedge C_{\text{in}}) \vee (\neg A \wedge B \wedge \neg C_{\text{in}}) \vee (A \wedge \neg B \wedge \neg C_{\text{in}}) \\ &\quad \vee (A \wedge B \wedge C_{\text{in}}), \qquad (1.1) \\ C_{\text{out}} &= (\neg A \wedge B \wedge C_{\text{in}}) \vee (A \wedge \neg B \wedge C_{\text{in}}) \vee (A \wedge B \wedge \neg C_{\text{in}}) \vee (A \wedge B \wedge C_{\text{in}}). \end{aligned}$$

Die Oder-Verknüpfung wird auch als **Disjunktion** und die Und-Verknüpfung als **Konjunktion** bezeichnet. Die beiden Formeln liegen in disjunktiver Normalform vor. Dabei sind die geklammerten Teilformeln durch Disjunktionen verbunden, und in den Teilformeln, die Disjunktionsglieder heißen, dürfen die Literale nur mit Und verknüpft sein.

Wir haben einen **Volladdierer** beschrieben: Wir können nun stellenweise von rechts nach links die Summe zweier Zahlen unter Berücksichtigung von Überträgen bilden, siehe Abbildung 1.4.

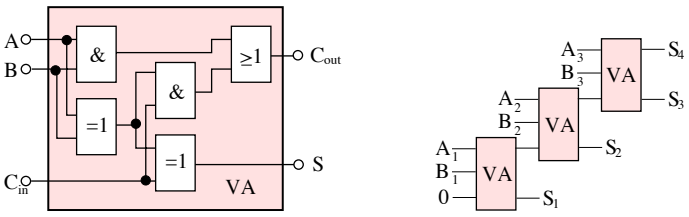

Abb. 1.4 Additionswerk mittels Volladdierer (VA): Ein VA ist über Gatter aufgebaut, die es als Hardwarebausteine gibt. Das &-Gatter realisiert $\wedge$, das $=$1-Gatter verknüpft die Eingänge mit dem exklusiven Oder $\oplus$, und das $\geq$ 1-Gatter implementiert das Oder $\vee$. Damit wird $S = (A \oplus B) \oplus C_{\text{in}}$ und $C_{\text{out}} = (A \wedge B) \vee ((A \oplus B) \wedge C_{\text{in}})$ berechnet. Vergewissern Sie sich, dass diese Formeln die gleichen Warheitswertetabellen besitzen wie die disjunktiven Normalformen (1.1).

Die Digitaltechnik beschäftigt sich mit weiteren Hardware-Komponenten wie dem Volladdierer. In der Praxis müssen Signallaufzeiten berücksichtigt werden, daher sind fast alle Baugruppen getaktet. In der Logik brauchen wir uns mit solchen Details nicht zu beschäftigen.

Bereits beim Volladdierer entstehen längliche Formeln. Wenn man diese in Hardware gießen möchte, ist es aus Kostengründen wichtig, diese möglichst weit zu vereinfachen, also zusammenzufassen. Auch Programme werden übersichtlicher, schneller und weniger fehleranfällig, wenn Berechnungen kurz und prägnant werden. Um kürzere Formeln zu erhalten, können wir mit